Projek Ilmiah Tahap Akhir I & II
Session 2003/2004

Implementation Traffic Control and Traffic Shaping in Linux using Web Approach

Name: Eric Ng Kang Yin
Matrix no: WEK010072
Supervisor: Mr. Nor Badrul Anuar bin Juma'at
Moderator: Mr. Phang Keat Keong

ABSTRACT

This report firstly would give an overview or an introduction to the world of Linux, and then explaining more on the networking capabilities of Linux, more specifically the traffic control.

Next, this report will review the essential knowledge behind the traffic control process in the literature review. These reviews include the operating systems, web servers, programming tools, and also a few existing systems concerning traffic control. Comparisons are made between the reviews mentioned above in order to make out which tools will be used to develop the system later on.

The third part will describe the methodology used to develop this system, together with their explanations. The fourth part, which is the System Analysis, will list out the requirements of the system in order to continue to the next phase of the system development.

Finally part 5 will be the system design where the flow of the system will be discussed and understood before embarking on the implementation phase. Following that would be system testing and system evaluation References are also included in order to clarify the details mentioned in this report.

ACKNOWLEDGEMENT

Prior to completing this report, I would like to express my sincere appreciation to my supervisor, Mr. Nor Badrul Anuar bin Juma'at. He has given me useful advice as well as guidance during the time of researching for this project. He has been a great guide while I was researching this system.

Secondly, I am also grateful to my moderator, Mr. Phang Keat Keong for his comments and remarks about the system that helps me to further perfect this system.

Lastly, I would like to thank all the course mates and friends who have had a helping hand in completing this research of the Traffic Control system for Linux.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

Chapter One - Introduction

1.1 *Overview of Traffic Control application for Linux*

For the past few years Linux has become immensely popular. It has grown from fewer from 100,000 users to more than 7,000,000 users in the span of 5 years [1] and this number is expected to increase in the coming years. This sudden growth of interest in Linux might be attributed partly to an anti-Microsoft bias, but the obvious reason most people use Linux is because Linux is far more powerful than any other operating system for a typical desktop personal computer. What was once a hacker's paradise has now become a stable and talented operating system, which makes Linux a more interesting alternative operating system to use. Best of all, Linux offers a UNIX environment at almost no cost at all which is exceptionally stable and leverages the strength of UNIX to the desktop environment.

Because Linux is somewhat a clone of the UNIX operating system, it inherits the latter's exceptional networking capabilities. Linux could do almost everything related to networking, including routing capabilities, packet forwarding, and as the title of this thesis states, traffic control. Traffic control is one of the advanced networking features in the UNIX environment where it includes bandwidth provisioning and also supports various methods for classifying, prioritising, sharing and limiting both inbound and outbound traffic.

This traffic control feature in Linux is controlled using the package **tc**, which is available in most of the Linux distributions such as Red Hat, Slackware, and Mandrake. Inside this package there are a few specific set of tools provided to handle traffic control in Linux efficiently. Below are the names of the tools in **tc** together with their brief explanations.

ip - a tool for viewing and configuring network devices, ip information, routes, rules, tunnelling options, and also route monitoring.

qdisc - deals with queuing, classes (sections of a class-based queuing discipline) and also filters (a way to put packets into queues)

The traffic control features in Linux leans heavily on Quality of Service (QoS), where these two features are mainly about limiting the amount of data leaving on a network interface. The limiting of data could pave some space for another type of data or to prevent misbehaviour. It is also worth noting that controlling outbound traffic is easier than controlling inbound traffic, which means that this paper will focus more on managing the outbound data being sent to the outer network.

One important issue to mention in traffic control is the **qdisc** or queuing disciplines for bandwidth management in a network. The **qdisc** is already part of the **tc** tool in iproute2. The keyword here is queuing, which translates to how we determine the way in which a data is *SENT*. With the way the Internet works, we have no direct control over the data other people send to us. As the Internet's main connection protocol is TCP/IP, there are fortunately some help from this protocol that would aid us in managing bandwidth in a network.

TCP/IP has no way of knowing the capacity of the network between two hosts, so it will just keep sending data faster and faster ("TCP/IP has a slow start") and when some packets start getting lost due to the lack of space to send them, it will slow down. In order to solve this problem, the "shaping" of the traffic is done to keep the bandwidth under control.

There are lots of different types of queuing disciplines, which can be divided into two main subs, those queuing disciplines with classes and those without. Classless queues differ from class queues because it has no internal subdivisions that are

configurable while the class queues may have many queues, which each are internal to the qdisc. A class may in turn have several classes under it, so a class can have a qdisc as a parent or as another class.

This thesis paper plans to utilize these tools in Iproute2 for controlling traffic in a Linux machine through online web pages. To be able to accomplish this task, this application will mostly be using the PHP language and also the Apache web server. The main point here is that the tools in Iproute2 (mainly tc and ip) will be configured via an online web page, from which these configurations will be set or manipulated to suit the network administrator's needs.

1.2 *The System's Objectives*:

- Able to control and manipulate the tools in Iproute2 through online web pages
- Able to control and monitor traffic going outbound and inbound from a device in the network
- Able to produce accurate information about the traffic in the network to the network administrator
- Able to produce a report of the network traffic to the network administrator

1.3 *The Scope of the System*

TCP/IP is practically the primary network protocol in all UNIX and Linux systems. Other protocols can also be used with Linux, but TCP/IP is the most widely used because it is included in all of the Linux distributions and it is the default protocol. TCP/IP is also the protocol of the Internet, which undoubtedly also adds to its popularity.

As TCP/IP is normally used over Ethernet networks, the Traffic Control application will apply to the Local Area Network topologies, where Ethernet is the main network design of LANs. This is not to imply that the traffic control could not be done on Metropolitan Area Networks (MAN) or Wide Area Networks (WAN) but because it is more appropriate because most of the Ethernet Network Interface Cards work cleanly under Linux. Since this application is for the LAN topology, the system will focus on controlling the outbound and also inbound traffic from the outer networks.

1.4 *Targeted Users*

The system will target the more technical kind of user, most suitably to be used by the network administrator of a Local Area Network. This is because he or she deals with the network directly more than any other administrators in a LAN.

The terms to be utilized in this proposed system is more inclined to the technical side, meaning networking terms and concepts. Therefore this system is not for use for the general user as they may not understand the terms and concepts applied to the system and thus could not utilize the system properly.

1.5 *Expected Outcome*

The system would be able to fulfil all of the aforementioned objectives and at the same time meeting the users expectations of the system. The system also will be able to help in the decision-making process of the network administrator: such as determining the suitable queuing discipline to implement in the network or finding a solution to network problems (congestion for example).

The system will generate reports (graphs) for analytical purposes and consecutively giving a clearer picture on the current network's status. Reports are available in daily, monthly and also yearly formats, whichever suites the needs of the network administrator.

1.6 *Project Timeline*

Below is the timeline of the entire software development process.

ID	Task Name	Start	Finish	Duration
1	Research Proposal	6/13/2003	6/23/2003	7d
2	Literature Review	6/24/2003	7/21/2003	20d
3	Requirement Analysis	7/22/2003	8/8/2003	14d
4	System Design	8/11/2003	9/1/2003	16d
5	Viva 1	9/2/2003	9/19/2003	14d
6	System Coding	9/22/2003	2/12/2004	104d
7	System Testing	12/1/2003	2/12/2004	54d
8	Documentation	6/12/2003	2/12/2004	176d
9	Viva 2	1/19/2004	2/11/2004	18d

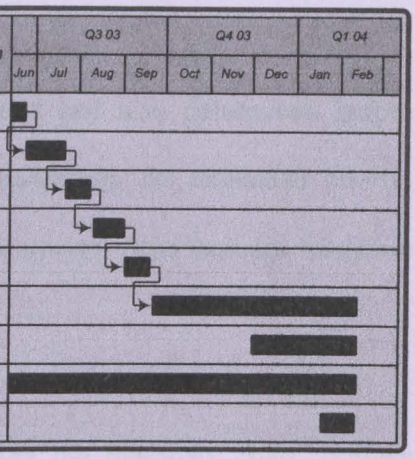


Figure 1.1 – Project Timeline for Traffic Control System

1.7 *Report Layout*

The purpose of this report layout is to give an early overview of the chapters in this report, which will be included during the development of this project. The report layout is as follows:

Chapter 1 – Introduction

This chapter provides the overview of the entire project,; project definition, project scope, project timeline that specifies the activities that needs to be done and also the expected outcome of the project.

Chapter 2 – Literature Review

The chapter here reviews the topics and studies it in detail so as to help in the research of Traffic Control in Linux. The methods used by traffic control will be discussed here in order to design the system that I am going to develop.

Chapter 3 – System Methodology

This chapter discusses the main methodology used prior to developing the system.

Chapter 4 – System Analysis

This is the chapter where requirements are defined and also determined before proceeding to the system design phase. The requirements are explained briefly, which includes functional and non-functional requirements. Also includes hardware requirements for developing the system.

Chapter 5 – System Design

This chapter outlines the flow of the system using the hierarchy chart, context diagram and also Diagram 0 in order to make the system easier to implement later on.

Chapter 6 – System Implementation

This chapter gives an outline of the techniques and approaches used to implement the system. Sample coding is also provided to further explain the processes involved.

Chapter 7 – System Testing

Testing the system is as important as designing and implementing the system itself. This chapter will try to cover the testing procedures used to check for software failures.

Chapter 8 – System Evaluation

Last of all is the system evaluation. The strengths and weaknesses of the system are compared and explained.

LITERATURE REVIEW

Chapter 2 – Literature Review

The main purpose of this chapter is to find and analyse any related information about the system that we intend to develop. The gathering of information about the existing systems is crucial in understanding the product we are going to develop, as this would give us a clearer picture of the strength and weaknesses in the soon-to-be-developed system. Without gathering this information, the built system would not be able to perform up to par with the expected results and thus condemn the system to more bug fixes and patches. In short, it does not seem too useful to the end-users who might plan on using the system.

Traffic control was discussed briefly in the previous chapter (Introduction) – and it will be a base line throughout the entire project. Most of what I have written about traffic control in the first chapter will continue to be discussed further in the following chapters in order to clarify the facts concerning the project. In the following discussion, we will concentrate more on the main process of traffic control – Quality of Service (QoS) to be exact.

2.1 *Quality of Service*

Quality of Service is briefly introduced in the last chapter as a means of managing bandwidth in a network. Bandwidth management has become more and more important as people realize the power of controlling bandwidth would eventually lead to better utilization of the network as well as equality to all users who use the network. In this section, I will explain more about the basic functions of the implementation of QoS in Linux, which includes the three main building blocks, namely Queuing Discipline, Classes and also Filters / Policing [2].

But before I continue, I would like to explain a bit more on the need to have QoS applied in a network. Why then would you need a Quality of Service in your network management? Perhaps you are having some problems with the network, or in another case, maybe customer complaints have been more and more frequent. Anyway, before utilizing QoS into the network, you will have to be sure that the network is viable or sustainable. If it isn't, there is no point in having QoS because it doesn't really fit the problem (it is also out of the scope of this thesis paper). Figure 2.1 gives a clearer picture of the QoS, which also provides the framework for the development of integrated services [3] and differentiated services [4] support in Linux.

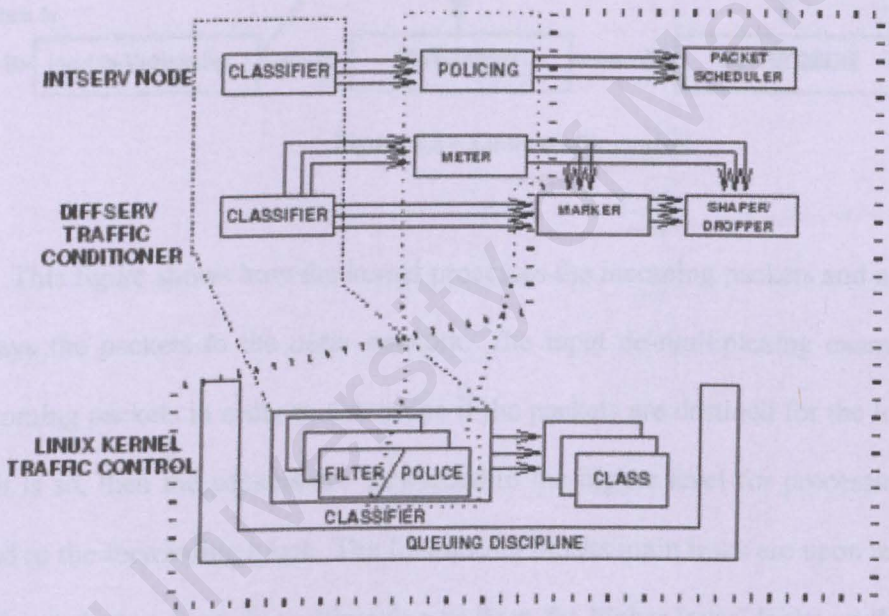


Figure 2.1 – Framework for developing "intserv" and "diffserv"

Proceeding on, I would like to mention first the rules that tc used for bandwidth specification [5]:-

*mbps = 1024 kbps = 1024 * 1024 bps => byte/s*

mbit = 1024 kbit = > kilobit/s

*mb = 1024 kb = 1024 * 1024 b => byte*

mbit = 1024 kbit => kilo bit

Internally the number is stored in bps and b. But when tc outputs the rate, it will use the following:-

*1 Mbit = 1024 Kbit = 1024 * 1024 bps => byte/s*

The basic implementation of the QoS in Linux [2] is shown in the figure below:-

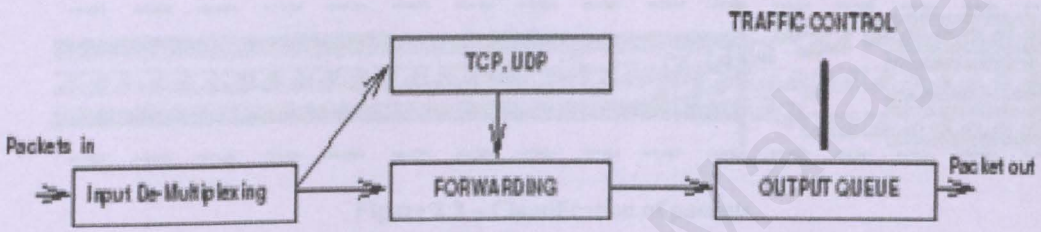


Figure 2.2 – Linux traffic control

This figure shows how the kernel processes the incoming packets and also how it relays the packets to the outer network. The input de-multiplexing examines each incoming packets in order to determine if the packets are destined for the local node. If it is so, then the packets are forwarded to the higher level for processing, else it send to the forwarding block, The forwarding blocks main tasks are upon receiving a packet (whether from de-multiplexing or from the higher layer) looks up the routing table and determines the next hop for the packet. After this step, the packets to be transmitted are queued on the output interface. It is at this point that the Linux traffic control comes into play. With this, Linux traffic control can be used to build complex combination of queuing disciplines, classes and filters that control the packets that are sent to the output interface.

To understand how to deal with traffic control in Linux, we have to think in terms of queues, or in other words pipelines, or a similar metaphor that represents a building up of data in or on a stack [6]. Consider a case where packets coming down from a pipe to be sent over a given network interface (going out onto the Internet, onto the internal network, etc.) all lined up and waiting to be sent. Think of these packets as belonging to sub-group like protocols (web browsing, file sharing, security) or computers (management, students) or any other classification.

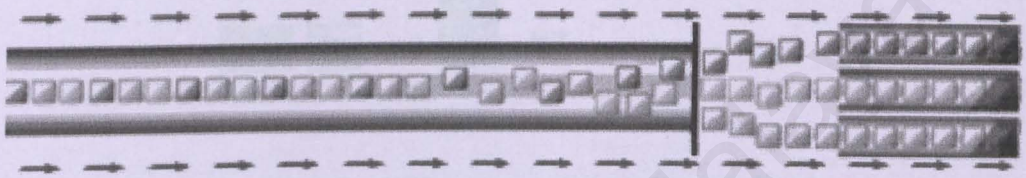


Figure 2.3 – Classification of packets

The above figure gives a brief self-exploratory example of the different classes of packets being sent out of the system to the outer network. The different colours indicate the different types of packet classes and the arrows show the direction of the packet flow.

In the next section, I will describe the three main building blocks of QoS as mentioned in the text above.

2.1.1 Queuing Discipline

Queuing Disciplines are algorithms that control how packets enqueued are treated [7]. It is also in fact the main building block for the support of QoS in Linux. One of the simple queuing disciplines would be FIFO (First In First Out). An example is shown below where the FIFO queue discipline consists of a queue of 16 packets.

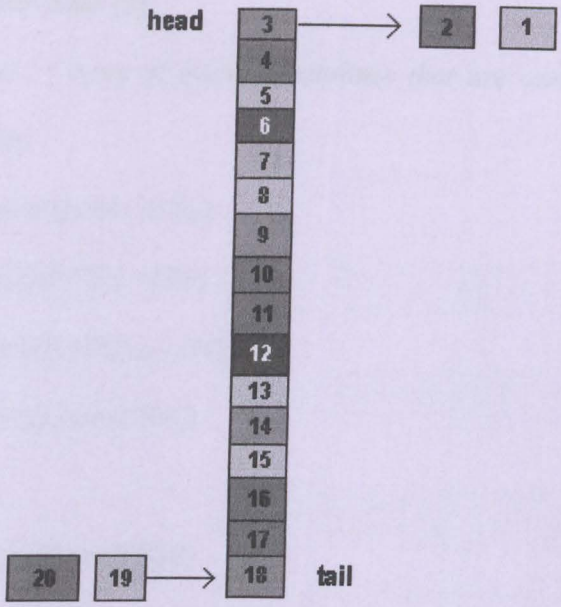


Figure 2.4 – Example of FIFO queue (colours indicate packet classes)

In this example, we would have to think our Linux box as a pure router, meaning that no packets will be sent to the upper layer and also no packets will be received from the upper layer (refer Figure 2.2). Apparently, the queue is moving 20 packets. Packets 1 and 2 had already entered and left the queue and packet 19 and 20 are in next in the queue to enter the queue. Now, before a packet can be accepted, another must be dispatched. If the packets arrive too fast, faster than they could be dispatched, then our router won't have any choice to even drop the packets.

This is bad for the network traffic that needs to conserve packets (protect TCP protocol behaviour and also applications that send/receive these packets). Thus we need to exercise complete control on how these packets are forwarded, how fast they are forwarded, which to delay, and which ones to drop. Practically, the queue is an area of memory on our router. If we expect packets of 1000-byte, we would have to reserve an area of 16KB on the router to implement this 16-packet FIFO queue.

As quoted by Saravanan [2]:

There are about 11 types of queue disciplines that are currently supported in Linux, which includes:

- *Class Based Queue (CBQ)*
- *Token Bucket Flow (TBF)*
- *Clark-Shenker-Zhang (CSZ)*
- *First In First Out (FIFO)*
- *Priority*
- *Traffic Equalizer (TEQL)*
- *Stochastic Fair Queuing (SFQ)*
- *Asynchronous Transfer Mode (ATM)*
- *Random Early Detection (RED)*
- *Generalised RED (GRED)*
- *Diff-Serv Marker (DS_MARK)*

Queues are identified by a handle <major number-minor number>, where the minor number is zero for queues. Handles are used to associate classes to queuing disciplines.

There are actually a lot more than those listed above, but it should be sufficient to explain the functions of queuing discipline. The list of supported queuing disciplines is really quite a lot as quoted by Saravanan [2], but we would not be reviewing all of them. In fact, we would be discussing briefly just two of them mainly TBF and CBQ. The TBF is an example of a classless queuing discipline and the CBQ is the one with classes.

But before that, let's take a look at these two figures below:

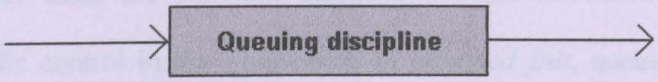


Figure 2.5 – Queuing Discipline

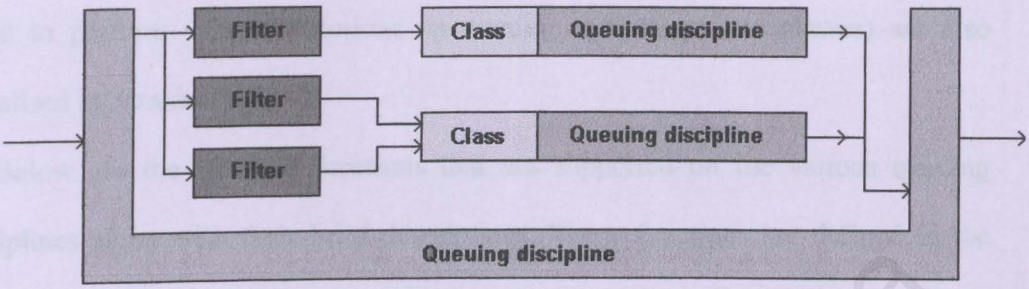


Figure 2.6 – Internal structure of Queuing Discipline

Figure 2.6 is just showing what was really happening inside the queuing discipline process. As you can see, the queuing discipline is tied with classes as the presence of classes and their semantics are fundamental properties of the queuing discipline [8]. Filters too can be arbitrarily combined with queuing disciplines and classes, as long as queuing disciplines have classes. From the list of supported queuing disciplines above, not all of them support classes. For example, Token Bucket Flow (TBF) does not have any classes attached to it.

One of the main advantages of QOS in Linux is the flexibility, which the combination of queues and classes that can be set up. Talking more about classes (they will be discussed more in the next section), they don't store the packets themselves but instead use another queuing discipline. These queuing disciplines may also have a number of classes and so on. It is from this fact that makes QoS support in Linux unique.

(Taking information from Saravanan [2]): Going to more technical matters, when a Linux kernel is configured for QoS support is booted up, the function *net_dev_init*

(in net/core/dev.c) calls the function pktsched_init (in net/sched/sch_api.c) to initialise the traffic control in the Linux box. In *pktsched_init*, queuing disciplines that have been compiled into the kernel are all registered and initialised. The pointers to access functions *tc_ctl_qdisc*, *tc_dump_qdisc*, *tc_ctl_class*, and *tc_dump_tclass* (used to perform various functions on queuing disciplines and classes) are also initialised in *pktsched_init*.

Below are the table of functions that are supported on the various queuing disciplines along with their brief descriptions. These functions are defined in the Qdisc_ops structure in include/net/pkt_sched.h.

Functions	Description
Enqueue	➢ Enqueues a packet with the queuing discipline ➢ The filters are run one by one until a match occurs. Once a match occurs, the enqueue function of the queuing discipline "owned" by the class is executed
Dequeue	➢ Dequeues a packet for sending ➢ Returns the next packet that needs to be sent out on the output interface ➢ This packet is determined by the scheduler in the queuing discipline
Requeue	➢ Requeues a packet for transmission ➢ After dequeing, if packet is not transmitted, the packet would be put back in the queue at the same position where it is dequeued ➢ Reasons for unable to transmit packets include: device could not establish its busy status before start of transmission, device is

	buggy, and fastroute option is enabled
Drop	➢ Drops a packet from the queue ➢ Simply involves dequeuing the packet from the queue and freeing the memory occupied by it
Init	➢ Initialise and configure the parameters of queuing discipline when it is created ➢ This function can be passed the arguments that will be used to configure the queuing discipline
Reset	➢ Reset the queuing discipline to its initial state (clears all queuing disciplines, stopping timers, etc.) ➢ The classes of these queuing disciplines are also automatically reset
Destroy	➢ Used to remove the queuing discipline ➢ Removes all classes and filters associated with queuing discipline ➢ Implemented in the *qdisc_destroy* function
Dump	➢ This function is used to dump diagnostic data associated with a queuing discipline ➢ Each queuing discipline maintains different diagnostic data that is dumped when the dump function is invoked

Table 2.1 – Functions supported on various queuing disciplines

I would now discuss about the Token Bucket Flow (TBF) queuing discipline. TBF is a simple qdisc that only passes packets arriving at a rate, which is not exceeding some administratively set rate, but with the possibility to allow short bursts in excess of this rate [5].

TBF is very precise, network and processor friendly. It would be best to choose this to just slow an interface down.

For this explanation I would refer to Ian's Token Bucket Example [9]. Token Bucket is described by two parameters, B (token depth) and R (token arrival rate). There is also another additional parameter, which is C (link capacity). The B parameter is a buffer bucket) that is constantly filled by virtual pieces of information (tokens) and these tokens arrive at a specific rate (which is R, the token arrival rate). Actually the most important parameter of the bucket is its size, the number of tokens it can store.

Each arriving token collects one incoming data packet from the data queue and is then deleted from the bucket. Associating this algorithm with the two flows (token and data) we have three different scenarios:

a) Data arrives in TBF at an **equal** rate of incoming tokens, meaning each incoming packet has its matching token and passes the queue without any delay.

b) Data arrives in TBF **smaller** than the token rate, meaning only a part of the tokens is deleted at output of each data packet that is sent out the queue (tokens accumulate up to the buckets size). This means the unused tokens can then be used to send data at a speed that's exceeding the standard token rate (short bursts of data).

c) Data arrives in TBF **bigger** than the token rate, meaning the bucket will soon be devoid of tokens, causing the TBF to throttle itself for a while. If packets continue to enter, the packets will start to be dropped.

The last scenario is very important because it allows shaping the bandwidth available to data that's passing the filter. Accumulation of tokens allows a short burst of overlimit data to be passed without loss, but any lasting overload will cause packets to be constantly delayed and dropped [5].

Next up would be Class Based Queuing (CBQ). This explanation about the CBQ will be discussed with reference to the people at LARTC [5], who have indeed made life easier for those inclined on learning more about the advanced networking features of Linux. CBQ as it seems to be, is a famous queuing discipline available, the most hyped, the least understood, and probably the trickiest to get it right.

Aside from being a classful queuing discipline, the CBQ is also a shaper. In this department the CBQ has some problem, which is to shape a 10Mbit/s connection to 1Mbit/s, the link should be 90% idle all the time. If it isn't, then we need to throttle so that it works 90% all the time.

Another difficulty of using CBQ is that CBQ derives the idle time from the number of microseconds that elapse between requests from the hardware layer for more data. When this is combined, we can speculate how full or empty the link is. This kind of measurement is not accurate enough: what if the link (100Mbit/s) is not really 100Mbit/s, due to some device problems. Then how do we find out the idle time?

Besides shaping, in CBQ, classes can have differing priorities and that lower priority numbers will be polled before higher priority ones. Each time a packet

requested by the hardware layer to be sent out the network, a weighted round robin process (WRR) starts, beginning with the lower priority ones These are then grouped together and queried to see if they contain any data. If so, it is returned. After a class has been allowed to dequeue a number of bytes, the next class within that priority is tried.

2.1.2 Classes

This section helps to explain more about the usage of classes in the queuing discipline. For this section, I would be referring to two references, mainly Saravanan [2] and also Babcock [6]. Previously (Section 2.1.1), I mentioned that queues and classes are irrevocably tied to one another.

From Saravanan's [2] quote: *Each class owns a queue, which by default is a FIFO queue. When the enqueue function of a queuing discipline is called, the queuing discipline applies the filters to determine the classes to which the packet belongs. It then calls the enqueue function of the queuing discipline that is owned by this class.*

This figure below taken from Werner [8], explains a lot about what is written by Saravanan about classes.

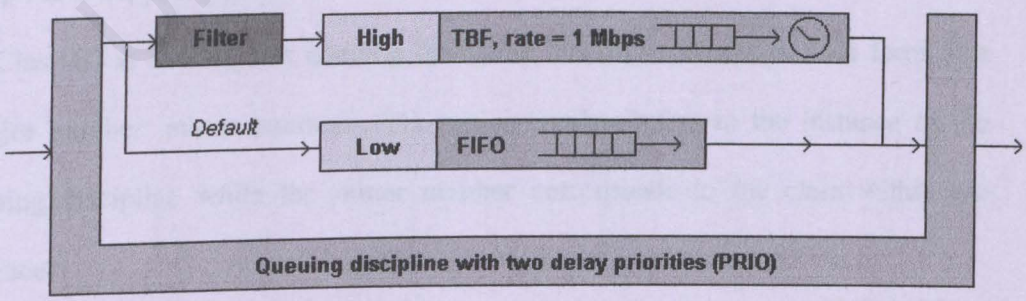


Figure 2.7 – Example of Queuing Discipline

It is an example of a queuing discipline with two different priorities; the filter will select the packets that will go to the higher priority class and relegate all the other packets to the lower priority class. The queuing discipline under each class then precedes to enqueue the packets accordingly.

Let's look at Babcock's [6] explanation: *Classes are sections of a class-based queuing discipline. CBQ, for example, allows multiple classes of packets to be defined which are then handled as a group by the root class. You can think of classes as being protocols, machines, or otherwise inter-associated sets of packets. For example, you may have only one home computer with a slow dial-up connection to the Internet. If you want your DNS lookups to have a higher priority than the rest of your network traffic, you could set up your DNS packets as one class, and the rest of the traffic as a second class. The actual assigning of packets to a given class is a separate task, but needs to be considered as one structure one's classes.*

This explanation by Babcock [6] further enhances Saravanan's [2] statement. Classes are simply said; an associated sets of packets that utilize a queuing discipline to control the packet flow. Classes work under a queuing discipline where under the class, there would be another queuing discipline tied to it. To be exact, not all of the queuing disciplines mentioned above support classes; only CBQ, DS_MARK, CSZ and p-FIFO support classes.

Class ID is similar to a queuing discipline identifier, structured in a form of a <major number: minor number>. The major number refers to the instance of the queuing discipline while the minor number corresponds to the class within the instance.

The table below lists the functions that can be performed on classes:

Functions	Description
Graft	➢ Functions to attach a new queuing discipline to a class ➢ The default queuing discipline is FIFO; in order to change this, a graft operation is performed on the class
Get	➢ Returns the internal ID of a class, given its class ID ➢ Increments the usage count of the class
Put	➢ Invoked when a class previously using the *get* function is de-referenced ➢ Decrements the usage count of the class ➢ If count reaches zero, may remove the class itself
Change	➢ Modify the properties associated with a class ➢ Can also create classes
Delete	➢ Delete the specified class ➢ Determines the usage of the class (checking reference count; if zero, deactivates and removes the class)
Walk	➢ Used to iterate over all the classes of a queuing discipline and invokes a call-back function for each of the classes ➢ This is to obtain diagnostic data for all classes of a queuing discipline ➢ This function is invoked when a *dump* request is made on the class
Tcf_chain	➢ Return the anchor to the list of filters that are associated with a class ➢ Each class is associated with a filter that are used to identify the packets that belong to a particular class

Bind_tcf	➢ Attach an instance of a filter to a class ➢ Increments the filter count for the class (similar to *get*) ➢ Class that is pointed to by filters cannot be deleted without first deleting the filters (queuing discipline refuses requests to delete a class if it is still in use)
Unbind_tcf	➢ Remove an instance of a filter attached to a class ➢ Decrements the filter count (similar to *put*) ➢ In order to delete class, the filter count must be zero
Dump_class	➢ Dump diagnostic data about the class ➢ Observing the data that is maintained on classes

Table 2.2 – Functions available for classes

In the next section, I will discuss the functions of filters in a queuing discipline and their role in the QoS in Linux.

2.1.3 Filters

Filters, in a general term means the process of filtering out the items to matched set of generated rules and policies. In traffic control (more specifically, queuing discipline), the filters are used to classify packets (filter) based on certain properties of the packet [2]. The properties would include anything from TOS byte in the IP header, IP addresses, port numbers, routing table entry, firewall marker and other specialised filters. This filtering process happens during the enqueue operation of the queuing discipline [8].

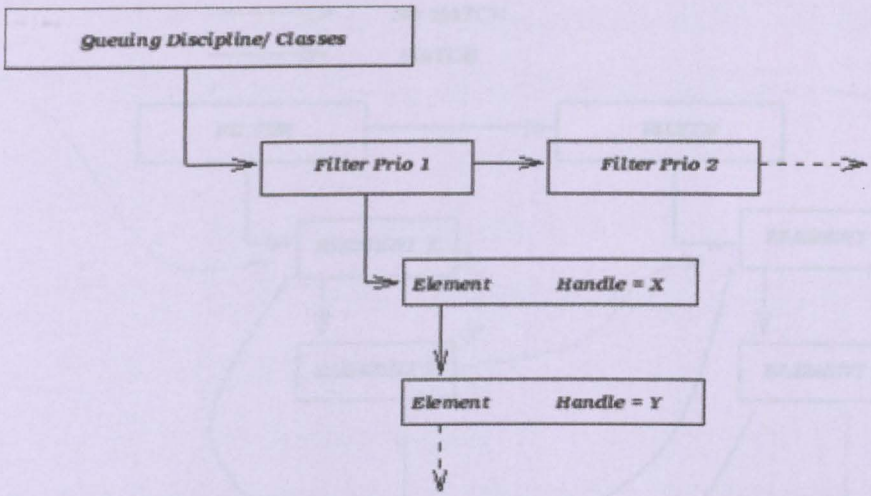


Figure 2.8 – Structure of Filters

As Figure 2.8 shows, the packets are filtered according to their priorities (ascending order). Later these packets are then controlled by the internal elements, which are referenced by 32-bit handles [8]. Handles are similar to class IDs, but the difference is that it does not split itself into major and minor numbers (handle 0 always refer to the filter itself). The internal IDs of these filters can be obtained using the *get* function as mentioned in the previous section of classes.

The process of filtering is explained below:

Once the protocol to which the packet belongs to is identified, filters that correspond to this protocol are applied in the order of priority. Inside each filter, the elements are all passed through in an attempt to classify the packet. Once classified, the enqueue function of the queuing discipline owned by the class is invoked. Figure 2.9 shows the matching process of the packets going through the filters.

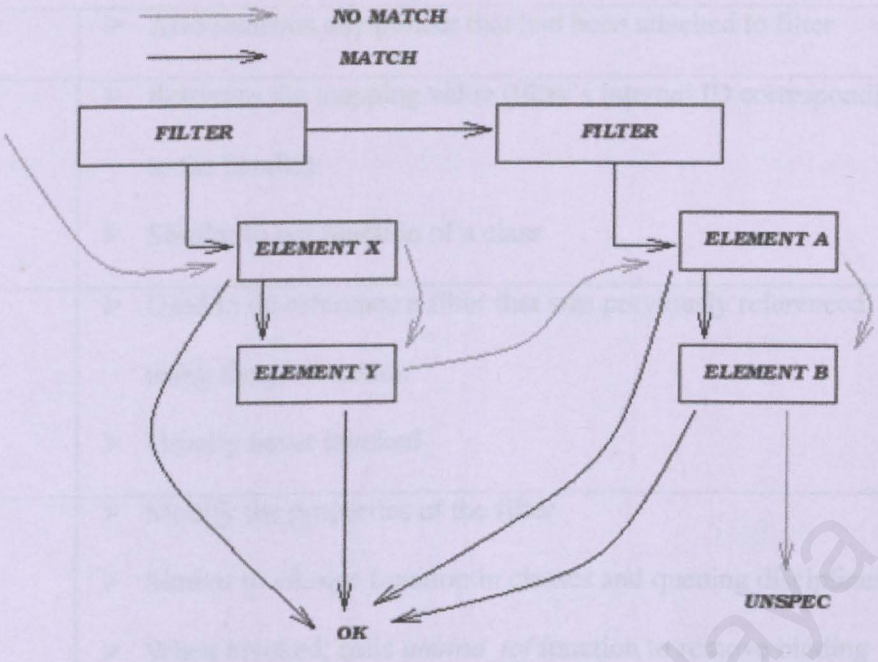


Figure 2.9 – Matching a Filter

As before, the functions supported in filters are as follows:

Functions	Description
Classify	➢ Matching a packet to a class based on certain properties of the packet ➢ Result of any classification can be any of four return values from policing, namely TC_POLICE_UNSPEC, TC_POLICE_OK, TC_POLICE_RECLASSIFY OR TC_POLICE_SHOT.
Init	➢ Initialises the parameters for a filter
Destroy	➢ Removes a filter ➢ If filters are also registered to classes, the *destroy* function will call the *unbind_tcf* function to de-register from theses classes

	➢ Also removes any policer that had been attached to filter
Get	➢ Retrieves the mapping value (filter's internal ID corresponding to the handle) ➢ Similar to *get* function of a class
Put	➢ Used to de-reference a filter that was previously referenced using the *get* function ➢ Usually never invoked
Change	➢ Modify the properties of the filter ➢ Similar to *change* function in classes and queuing disciplines ➢ When invoked, calls *unbind_tcf* function to remove binding between class and filter. Then calls *bind_tcf* to bind the filter with the new properties to the class
Delete	➢ Deletes a particular element of a filter ➢ Similar to the *change* function above, *unbind_tcf* is invoked on the class to which element is attached ➢ Policers attached to the element are also removed
Walk	➢ Iterate over all of the elements of a filter and invokes a callback function for each of the elements ➢ Usually used to obtain diagnostic data for all elements of a filter
Dump	➢ Dump diagnostic data about the filter and one or more of its elements

Table 2.3 – Functions available for filters

There are two types of filters, generic filters and specific filters. They vary in the scope of packets their instances can classify: generic filters use one instance per queuing discipline to classify packets for all classes. This is illustrated in Figure 2.10.

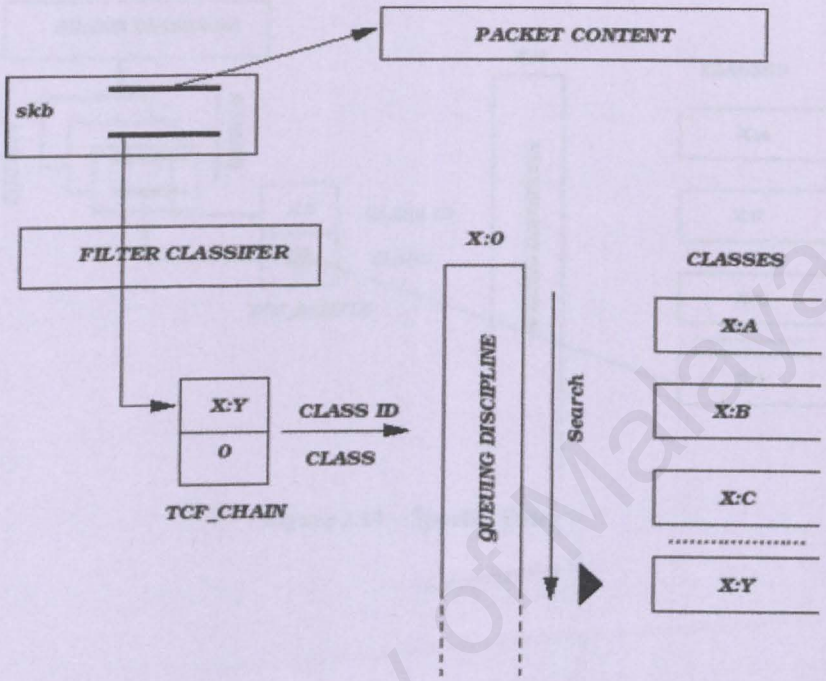


Figure 2.10 – Generic Filter

On the other hand, specific filters needs more than one instance or its internal elements per class in order to identify packets belonging to this class. An internal filter ID distinguishes multiple instances of a specific filter, which is similar to the internal ID used by classes. Since the specific filters have at least one instance of the filter per class, the internal ID can be stored in the *tcf_proto* structure, which enables a fast lookup for the class [2]. The figure below illustrates the specific filter:

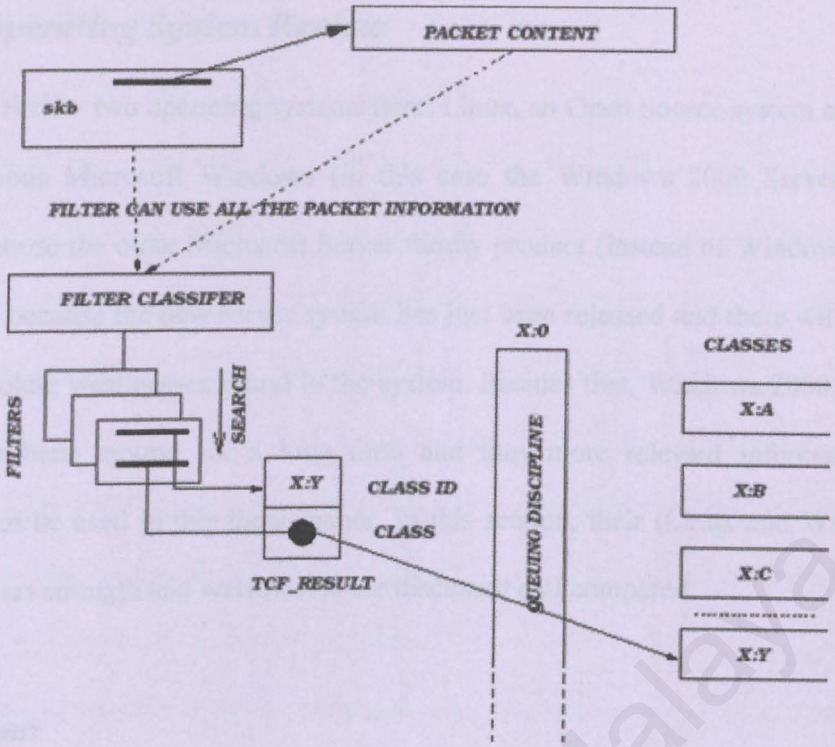


Figure 2.11 – Specific Filter

2.2 *Operating System Review*

I will review two operating systems here: Linux, an Open Source system and also the infamous Microsoft Windows (in this case the Windows 2000 Server). The reason I chose the older Microsoft Server family product (instead of Windows 2003 Server) is because the new server system has just been released and there will likely be no absolute weaknesses found in the system. Besides that, Windows 2000 Server is already been around for a long time and thus more relevant information is available to be used in this thesis paper. In this section, their (Linux and Windows 2000 Server) strength and weaknesses are discussed and compared.

2.2.1 Linux

This paper generally talks Linux more than Microsoft Windows, as it is, anyway the title of this thesis paper. Linux, as was explained, albeit briefly in the first chapter a free operating system with a host of neat features. Best of all, this comes at virtually no cost to the consumer, unlike other proprietary operating systems that require the end-user to pay for the license of the operating system before using it. I will not discuss more about Linux aside from their strength and weaknesses. They are listed below:

* **Note:** most of these facts are taken off from the Simple End User Linux (SEUL) Project's website.

Strengths (Benefits) of the Linux operating system **[10]**

- *Linux is network friendly*

  Linux was developed by programmers over the Internet, so its networking capabilities and features are given high priority. Linux is capable of acting as client and/or server to any of the popular operating systems in use today, and is also quite capable of being used as a Internet Service Provider. Linux supports most of the major protocols, and some of the minor ones are also supported. With support for Simple Network Management Protocol and also other services such as Domain Name Service, Linux is good enough to serving large and complex networks.

- *Linux needs only minimal hardware configurations*

  Linux, though it seems to be a powerful and robust operating system, surprisingly need only minimal hardware configuration in order to run properly. It all depends on what services that are needed and the quantity, for example a small college in Southern Minnesota ran a 56K baud lease line (access for a student body of around 300 persons) gateway with e-mail, DNS, and FTP on a single Pentium 486/33, 32MB RAM and a lot of large SCSI disk drives.

- *Linux is Multi-User*

  Linux, a UNIX clone, which inherits the latter's design philosophy, meaning Linux is a full-fledged multi-user operating system. There are advantages using this feature, even if it is just one or two users who will be using it. The security (necessary for protecting sensitive information) is built into Linux at several levels. Linux is designed for multi-tasking, which makes it all the more powerful as an operating system.

- *Linux is Open*

  This is perhaps the strongest point in the Linux operating system advantages: open source. This open source means that the entire base system's source code is accessible and can be modified in any way the user wishes to unlike proprietary operating systems where the source code are inaccessible and thus users have to rely upon the support team in the proprietary operating system's company.

  The large numbers of software authors and beta testers are also a bonus: software refinement is much faster and better. The unavailability of corporate deadlines increases the software quality.

  End users benefit the most from open source as they can be more involved in the software development process. They also have the option of contacting the software developer if they have any questions about the software.

- *Linux is Free*

  Linux is "free" in two senses: the consumer is free to modify the software, and also the consumers can own Linux without paying anything at all. There are two ways of getting Linux, first by downloading from the Internet and second is getting a Linux distribution by CD-ROM. The former is time consuming but if the user's connection is fast enough, he or she need only pay for the connection costs. As for the latter, it would be a more viable choice for most users. This is because they can pay a few dollars (physical disk, shipping and tax charges) to get it and eases the wait for downloading the Linux system through the Internet.

- *Linux is Reliable*

  Linux is one the stable operating systems out in the market today: reasons for this are programmers were writing Linux for other programmers and not the corporate system. Therefore there is not much deadline pressure when one is writing programs as a hobby. Linux can actually run for days, or even years because the system can be upgraded and modified "on the fly" – the only reason to turn off a Linux system is when adding new hardware and installing a new kernel.

  Another reliable benefit of Linux is the availability of software packaging standards (by Red Hat and Debian). Both of these standards are easy to use because they will check for software dependencies before executing and also have the ability to upgrade the system without rebooting the system. Removing software is also fast, simple and easy.

- *Linux is Backwards Compatible*

  Linux have superb support for old hardware. In fact it is easier to find support in the 486 era instead of the latest hardware in the market now. This is due to the lack of driver being written for Linux.

  This means that users can reuse their old hardware (rendered obsolete by other operating systems) to an extent that it still as useful using Linux.

Weaknesses (Drawbacks) of the current Linux system [**11**]:

- *Less Choices of Software*

  Proprietary operating systems (Windows, MacOS, Unix and etc) have been around for a long time. Linux, on the other hand, has only been around since 1991. This makes the software choices a bit harder to find but as time goes

on, more and more packages are included into Linux such as the reliable StarOffice suite and powerful graphical desktop environments (KDE and GNOME). The only thing to do is to use Linux where it makes sense (use where it is necessary to use it).

- *Difficulties in Migrating from Current Systems*

  The conversion or migration from previous systems (usually proprietary operating systems) can be quite difficult to manage. This is because theses software vendors are adept at "lock-in" strategies that make it hard to switch to competitive offerings. In order to switch to Linux, a little investment is needed. In the case of a small business companies, these investments include converting existing documentation, data and others to Open Source software. Besides that, personnel are also required to undergo training, procedures and processes need to change in order to adapt to the new system and new maintenance and upgrade policies are needed.

- *Lack of Documentation*

  Though there has been some work being done on documentation for Linux, for example the Linux Documentation Project at www.tldp.org, generally there is still a lack of decent documentation especially for beginners using Linux. Most papers are still a bit on the technical side and this might put off new users from trying out Linux.

2.2.2 Microsoft Windows 2000 Server

The Windows 2000 Server has addressed most of the weaknesses of its predecessor the Windows NT Server. It was built upon NT technology and the interface of this system is similar to those of Windows 95/98. Below are listed the strengths and weaknesses concerning the Microsoft Windows 2000 Server:

Strengths (Benefits) of the Windows 2000 Server [12]:

- *The Vast amount of Support*

  Microsoft is a company that continuously provides support and updates to its end users. This ensures that the end users have the opportunity to have a place to turn to whenever a problem occurs. The documentation is also well written and updated often, thus users will have no trouble looking up the troubleshooting guide and find their answers there.

- *Ease of Use*

  The Microsoft Windows family is in fact the most widely used operating system in the world today. Thus if a new operating system (for example Windows 2000 Server) were put in the software market, the users who were familiar with Windows (who isn't?) systems would have no problem settling in.

- *Integration of multiple Microsoft developed software*

  Most people using Windows 2000 Sever hosting utilizes the Active Server Pages (ASP) technology. The integration of Microsoft developed software products have cleared the need to have multiple different software vendors in order to publish a web page on the Internet.

Weaknesses (Drawbacks) of the Windows 2000 Server:

- *Consume more Resources*

  The one drawback of Windows systems are that they consume more system resources than a Linux system. In order to run properly, the Windows 2000 Server requires a more powerful computer.

- *Higher Cost*

  As Windows is a proprietary operating system, we would have to spend more money to get the license to use the software. The license varies between the Windows family, but the Server series are especially expensive to own because of their custom made software designated for servers. Another issue that adds to this drawback is the distribution of licences to each of the workstations using Microsoft Windows. The weakness mentioned above also contributes to a higher cost, where you would have to spend more on buying a more powerful hardware to run Windows properly.

- *Reputation*

  Windows does not have the reputation of being a stable operating system. Instead they are famous for the amount of patches issued and also the infamous "blue screen of death" that happens whenever a system error occurs. Windows too have a reputation for system crashes, which might have the need to reinstall back the operating system.

- *Needs a special BIOS in order to run* **[13]**

  Windows 2000 Server requires a special BIOS (ACPI – Advanced Configuration and Power Interface) in order to run properly. This can cause problems for those with existing systems without this type of BIOS may have

to upgrade the BIOS, or in the worst case getting a new motherboard that includes an ACPI BIOS.

2.3 *Web Server Review*

Web servers are part and parcel of the Internet as we speak. The concepts of the web servers: web servers present a virtual hierarchy of information to browser programs [**19**]. This is actually the *path* part of the URL in the browser. The virtual hierarchy does not need to correspond directly to the physical organization of data. Servers can usually be configured to map parts of the virtual hierarchy to different areas of the physical file system, or redirecting them to other servers. The URL may also identify a program that returns information in the form of a document. Then, information in the URL after the part that identifies the program is passed to the program as data that may be evaluated to determine the content of the document generated.

The web servers that I will review in this section are Apache and also Microsoft's Internet Information Server (IIS).

2.3.1 Apache

Apache, the HTTP server, is actually one of the projects being developed by Apache Software Foundation. This Apache web server is designed to fulfil the goal of the project: provide secure, efficient, extensible server that provides HTTP services in sync with the current HTTP standards [**20**].

Apache is the most widely used web server on the Internet since April of 1997. The July 2003, Netcraft Web Server Survey found that 63% of the web sites on the Internet are using Apache.

Some of the features included in Apache are as follows:

- *DBM databases for authentication*

  Allows setting password-protected pages with enormous numbers of authorized users, without bogging down the server

- *Customized Responses to Errors and Problems*

  Allows setting up of files (even CGI scripts), which are returned by the server in response to errors and problems

- *Multiple DirectoryIndex directives*

  Able to use in*dex.html* and *index.cgi*, which instructs the server to either send back *index.html* or run *index.cgi* when a directory URL is requested, whichever it finds in the directory

- *Virtual Hosts*

  This feature, also known as multi-homed servers, allows the server to distinguish between requests made to different IP addresses or names (machine name). Also allows dynamically configurable mass-virtual hosting

- *Configurable Reliable Piped Logs*

  Allows setting Apache to generate logs in the format that you want. On most UNIX architectures, Apache can send log files to a pipe, allowing for log rotation, hit-filtering, real time splitting of multiple vhosts into separate logs, and asynchronous DNS resolving on the fly.

2.3.2 Internet information Server (IIS)

Internet Information Services (IIS) 6.0 is the latest addition to the Microsoft's Web server family. Designed for intranets, the Internet, and also the extranet, IIS enables organizations large or small to easily deploy powerful Web sites and

applications. In addition, IIS also provides a high-performance platform for applications build using the .NET Framework.

Some of the benefits of IIS are:

- *Reliability*

  Makes the .NET application more reliable with the fault-tolerant process architecture. Also ensures availability by using automatic worker process recycling to refresh applications.

- *Manageability*

  Easily edit XML format configuration data using text editor, and some of the ease of use feature include importing and exporting settings from the configuration data for administrative tasks to simplify deployment on multiple computers.

- *Scalability and Performance*

  Has improved capacity planning tracing for HTTP, ISAPI filters, ISAPI Extensions, ASP, and ASP.NET. The tracing data can then be used to diagnose delays and aid in the debugging process.

- *Security*

  Taking advantage of SSL to increase resistance to attack strategies with an architecture that isolates user code in worker processes. Automatic patching and a variety of authentication schemes are also available

I have reviewed a paper comparing the Apache web server together with Microsoft's IIS. This was conducted by Amisu Salam-Alada and Abdul Waheed of the Performance Engineering Laboratory of the Computer Engineering Department

of the College of Computer Science and Engineering, King Fahd University [22]. Below is the summarized version of the paper:

The tests were done using Webstone and compared Apache and IIS with the following performance metrics: **throughput** in terms of **bits/sec**, **average latency** of each transaction, **connection rate** and lastly **server CPU utilization**.

Throughput in terms of bits/sec

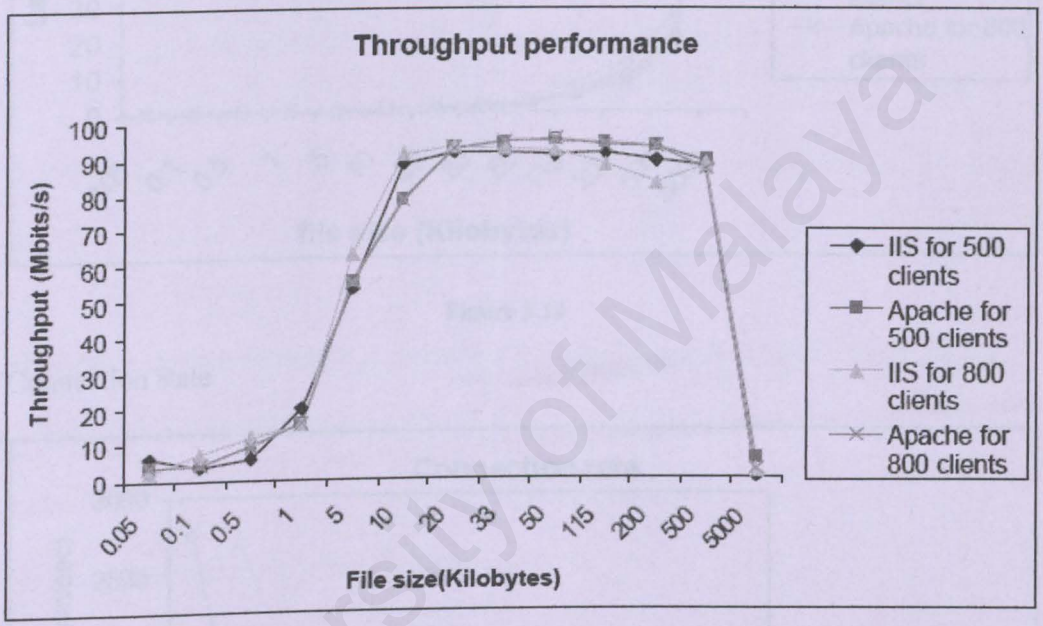


Figure 2.12

Average latency of each transaction

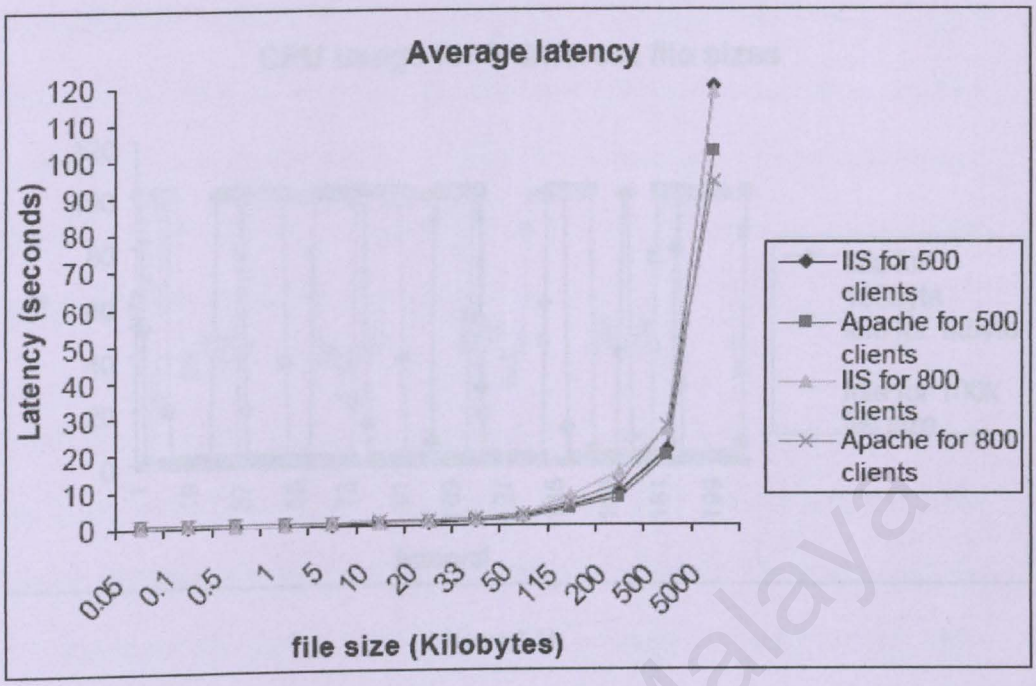


Figure 2.13

Connection Rate

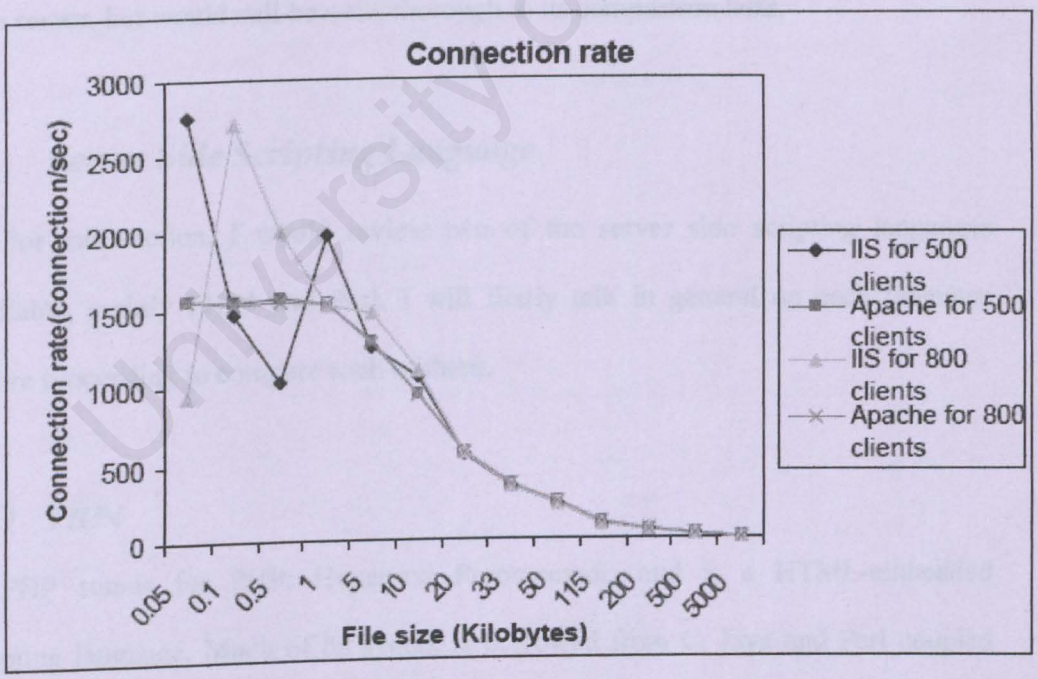


Figure 2.14

Server CPU utilization

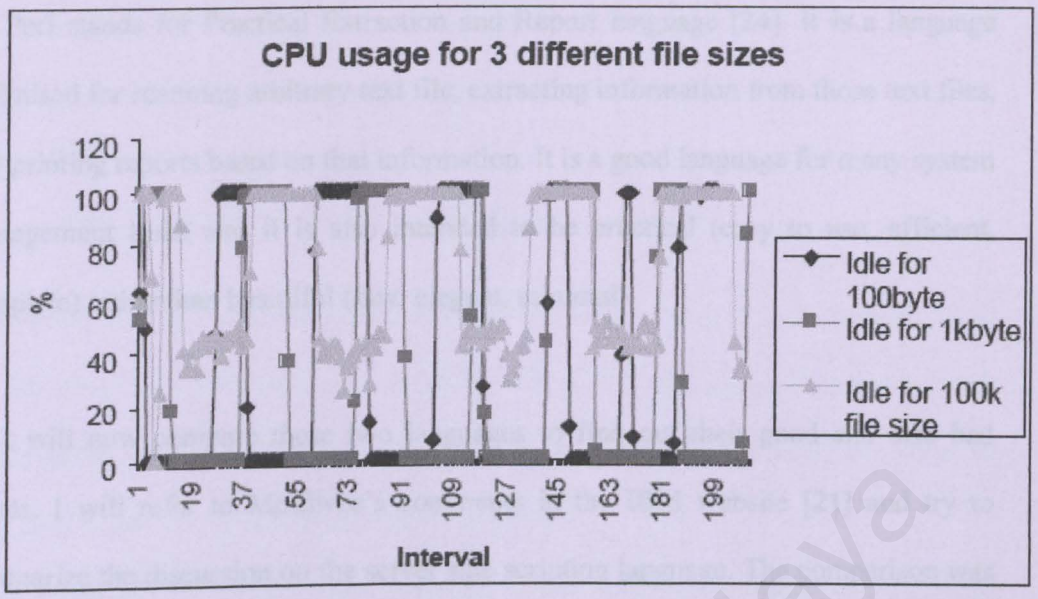


Figure 2.15

The above figures do little justice to the real performance of Apache and the IIS web server, but would still be quite thorough in its comparison tests.

2.4 *Server Side Scripting Language*

For this section, I would review two of the server side scripting languages available, mainly PHP4 and Perl. I will firstly talk in general on each language before proceeding to compare each of them.

2.4.1 PHP4

PHP stands for PHP: Hypertext Preprocessor, and is a HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl coupled together with some unique PHP-specific features thrown in. The goal of PHP is to allow web developers to write dynamically generated pages quickly and easily.

2.4.2 Perl

Perl stands for Practical Extraction and Report language [24]. It is a language optimised for scanning arbitrary text file, extracting information from those text files, and printing reports based on that information. It is a good language for many system management tasks and it is also intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

I will now compare these two languages to find out their good and also bad points. I will refer to McElwee's comments in the IBM website [21] and try to summarize the discussion on the server side scripting language. The comparison was made using tasks and seeing how the specific language manages the task using programming.

- *Task 1 – Putting form field data into variables*

  Perl

```
read(STDIN, $buffer, $ENV{'CONTENT_LENGTH'});
@pairs = split(/&/, $buffer);
```

  PHP

```
$data = $HTTP_POST_VARS["data"];
$data2 = $HTTP_POST_VARS["data2"];
```

- *Task 2 – Search and Replace*

  Perl

```
$data =~ s/cat/dog/gi;
```

  PHP

```
$data = eregi_replace("cat","dog",$data);
```

➢ *Task 3 – File Writing*

Perl

```
open (OUT, ">>file.txt");
```

PHP

```
$out = fopen("file.txt", "a");
```

For task 1, Perl's standard block of code isn't hidden away in a module. Here, all the "name=value" pairs are taken out of the string and each assigned to a slot in the @pairs array. As for PHP, it skips the first step and jump straight into assignment as long as you get the variable names that you expect.

For task 2, Perl has the most powerful regular expression engine and excels at text manipulation. AS for PHP, the interface appears awkward and convoluted when compared to Perl's elegant syntax.

For task 3, both of the languages are a little similar, where the codes are simple, intuitive and almost identical.

That said, choosing the right server side scripting language all depends on how you perceive it and for what purpose.

2.5 *Existing System Review*

This section will review some of the existing systems, which utilizes traffic control using online methods (configuring the network traffic through the Internet for instance). The sources are taken from the http://sourceforge.net site, where software developers host their software (all are open source software).

I will review only some of them, namely Dante, Traffic Discovery, and also Cricket.

2.5.1 Dante

Dante is the name of a system that was developed by Matt Critcher, a UNIX System Administrator in Lyon College of Information Services and Computer Science. The software's prime goal is to solve the bandwidth management problem in college campuses, where the students use peer-to-peer file-sharing programs on the WAN link.

The solutions to this matter are discussed in Critcher's paper [**20**]. The first one, adding another high-speed connection to the campus is not a good idea. Due to the costs of maintaining a high-speed Internet connection line, the idea of getting another line would seem to pile more pressure on the financial situation of the college (unless of course, money grows on trees).

After some consideration, it would be best to use a hardware-oriented solution. But then again, the costs of these intelligent devices need also be taken into consideration. So, this idea also could not seem to work. Finally after much thinking, the best solution is to fully take advantage of the amount of free software available in the Internet. Most (and nearly all) of the best system-level software in the last two decades was first developed on UNIX, which inevitably leads to Linux. Linux, which inherits UNIX's exceptional flexibility and stability had lots of networking related software to date and from this reason, a software solution is used to counter the problem above.

Onward to the details: Dante works by calling the help of Linux 2.4's internal QoS networking components, the tc program, and Ethernet bridging software. Dante is placed between the WAN link and the LAN. The traffic flows from one side of Dante's Ethernet Bridge, gets shaped by tc, and then sent out the other side of the bridge.

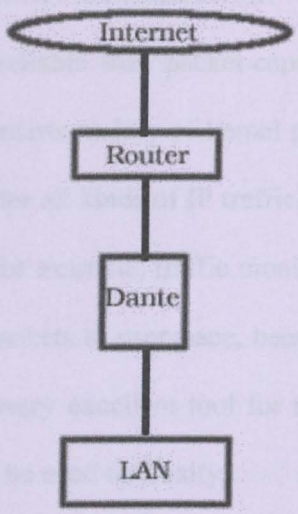


Figure 2.16 – Traffic flow (with Dante)

The summary of how Dante works is as follows:

- Create Ethernet bridge (creates a tunnel between two or more Ethernet interfaces)
- After the bridge is created, packet is classified by tc based on type by port number (uses the U32 classifier)
- After classification, packets are queued (CBQ). There is a minimal bandwidth that tc will maintain, so traffic will always flow.

Anyhow, Dante is not without flaws. The one potential problem is HTTP transfers (usually takes place on port 80 and 443 (HTTPS)), where in the case of conversations not taking place in those two ports. Then Dante is not able to classify it.

2.5.2 Traffic Discovery Linux Kernel Project

The main purpose of this project is to help system administrators to monitor different kinds of IP traffic on a Linux router **[23]**. It is supported for two kernel releases - 2.2.x (2.2.19) and 2.4.x (2.4.18).

The advantages of using this software are listed below:

- Much more faster and reliable than packet-capture-type traffic monitoring projects. This project consists mainly of kernel patch, which modifies Linux kernel to track and monitor all kinds of IP traffic. Packets do not leave kernel space for that purpose (for example, traffic monitors which user pcap library are obliged to transfer packets to userspace, because of pcap library nature). This makes traffic discovery excellent tool for routers under high pressure, where all resources must be used optimally;
- Provides complicated criteria for traffic monitoring;
- Easy to manage;
- Two modes of traffic monitoring available - route-based and host-based;
- Output information is parsed in file in process information pseudo-filesystem (/proc), where it is viewed as a table (ease to parse).

The disadvantages are:

- OS-specific;
- There is no user-friendly interface for management and statistics

2.5.3 Cricket

Cricket is a high performance, extremely flexible system for monitoring trends in time-series data [25]. Cricket was expressly developed to help network managers visualize and understand the traffic on their networks, but it can be used all kinds of other jobs, as well.

Cricket has two components, a collector and a grapher. The collector runs from cron every 5 minutes (or at a different rate, if you want), and stores data into a data

structure managed by RRDTool. Later, when you want to check on the data you have collected, you can use a web-based interface to view graphs of the data.

Cricket reads a set of config files called a config tree. The config tree expresses everything Cricket needs to know about the types of data to be collected, how to get it, and from which targets it should collect data. The config tree is designed to minimize redundant information, making it compact and easy to manage, and preventing silly mistakes from occurring due to copy-and-paste errors.

Cricket is written entirely in Perl and is distributed under the GNU General Public License.

METHADOLOGY & SYSTEM ANALYSIS

Chapter 3 – System Methodology

3.1 Methodology

Developing software requires good planning techniques besides the technical expertise wielded in order to finish the project in time, in schedule and also in budget. Thus said, it is best to come out with an effective methodology to tackle the planning of the software project. Methodology in general describes about a set of procedures that indicates which tasks to perform, when and how to perform them, and also how to manage the process to develop the most cost-effective and last but not least a high quality software system [**26**]. This methodology is crucial in guiding the developers through the entire life cycle of the software development to give everyone a common understanding for productively communicating and talking with one another.

With this in mind, the methodology that was chosen to develop this project is the Waterfall Model With Prototyping method. This model is actually a modification of the Waterfall model, which tries to correct the problems that had bogged the original modelling process. The term 'Waterfall' was not actually described by the author, Dr. Winston W. Royce but in fact it was coined by other researches as the model's designs were cascading from one phase to the other [**27**]. In order to understand the reason I chose this modified version of the Waterfall model, let's look at some of the drawbacks of the original Waterfall model. They are listed below:

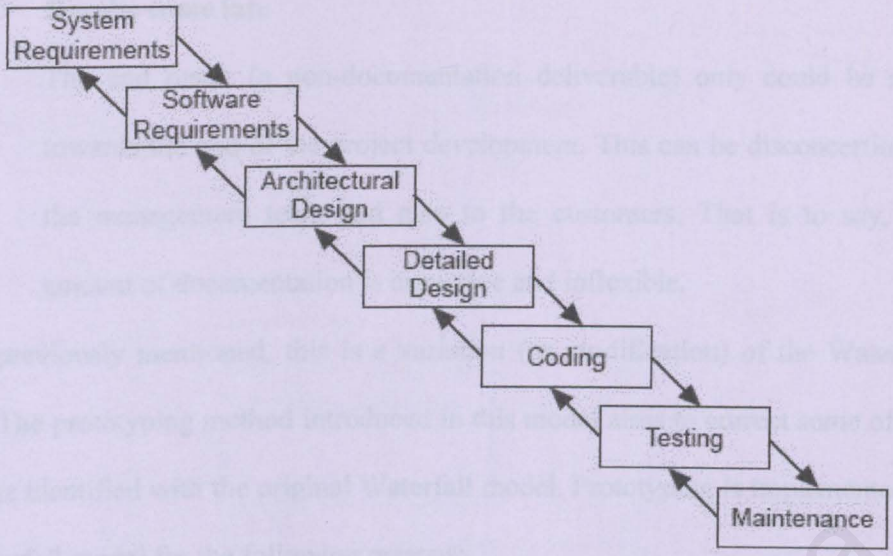


Figure 3.1 – The typical Waterfall Model

- **Inflexible**

  This model does not permit returning to an earlier phase [28] (but allows returning to the previous phase): *e.g.* returning to the requirements phase while at the design phase – refer to the Waterfall figure above. This actually requires a costly rework (each phase requires a formal review and extensive documentation development) of all the phases involved between the two development phases.

- **No overlapping between development stages**

  The lack of overlapping (the process of starting a stage within a stage) between development stages will cause errors in the long run: *e.g.* you may discover issues during the coding stages that point out the errors made during the requirements specifications stage. This also causes the project's development progress to slow as the stages do not overlap with one another.

- **Results come late**

  The end result (a non-documentation deliverable) only could be seen towards the end of the project development. This can be disconcerting to the management team and also to the customers. That is to say, the amount of documentation is excessive and inflexible.

As previously mentioned, this is a variation (or modification) of the Waterfall model. The prototyping method introduced in this model aims to correct some of the problems identified with the original Waterfall model. Prototyping is implemented in the Waterfall model for the following reasons:

- A prototype is practically a partially developed system, which would decide whether it is suitable to be used as the final product
- Prototypes are essential for verification and validation; verification ensures each function works accordingly while validation warrants that the system implements the entire requirement in the specification
- Prototypes can be useful for demonstrating how a design deals with a set of requirements. Prototypes can be built and tested several times until you have a clearer picture of your overall objectives. Besides clarifying the requirements, it also defines many areas of the design simultaneously.

Below is the figure of the modified Waterfall model, which incorporates prototyping into the development life cycle.

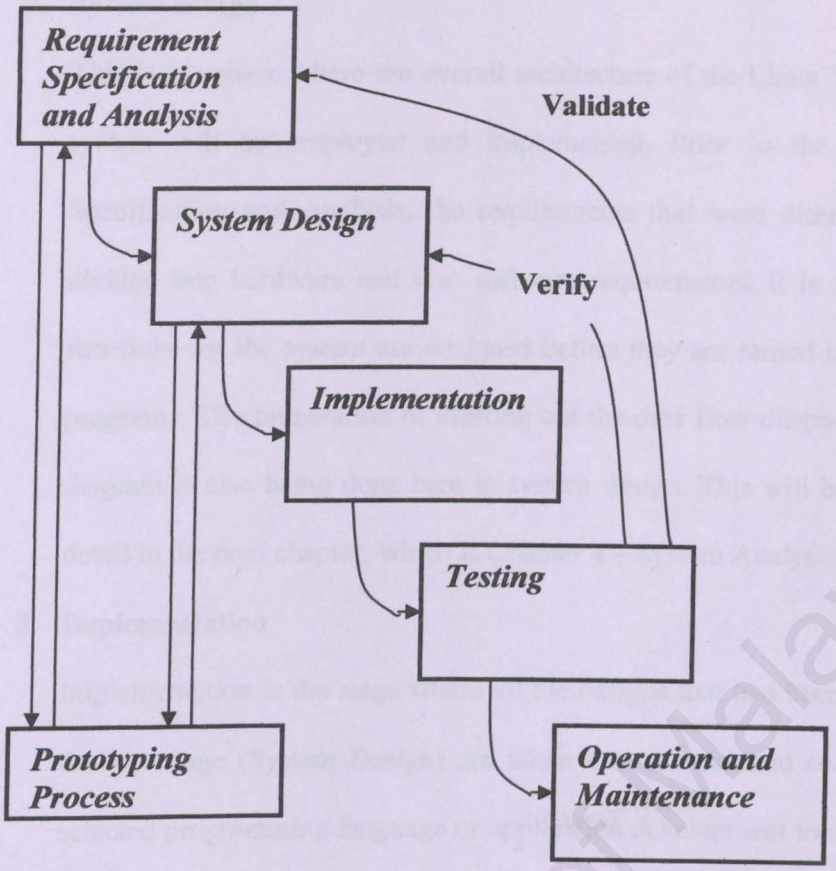


Figure 3.2 – Waterfall model with Prototyping

Referring to the above model, there are mainly five stages. The explanation for each of the stages follows:

1. **Requirements Specification and Analysis**

   This stage has already been completed prior to Chapter 2 where the information regarding this project is gathered. The information is garnered mostly through the Internet and books concerning traffic control and Linux whether in detail or in general. The main objective of this stage is to establish the system's requirements and goals. It is also necessary to identify and understand the aforementioned objectives of the system in order to proceed to the design phase.

2. **System Design**

This is the phase where the overall architecture of the Linux Traffic Control system will be employed and implemented. Prior to the Requirements Specification and Analysis, the requirements that were discovered will be divided into hardware and also software requirements. It is from here that functions for the system are designed before they are turned into executable programs. The preparation of drafting out the data flow diagram and context diagram is also being done here in system design. This will be discussed in detail in the next chapter, which is Chapter 4 – System Analysis.

3. **Implementation**

Implementation is the stage where all the designs that had been made during the last stage (System Design) are taken into account and coded using the selected programming language or application development tools specified in the System Design stage. Each function is to be tested extensively to verify that it is functioning according to specifications. In this stage also, various software bugs will be addressed and eliminated.

4. **System Testing and Implementation**

The modules that were developed separately (this also includes everything concerning the system) will be brought together and tested as an entire and complete system. This testing being done here aspires to make sure that the system meets the user's requirements, thereby ensuring the system's usefulness. Enhancements will also be introduced into the system if necessary to improve the system even further.

5. **Operation and Maintenance**

Maintenance is just as important as developing the system itself. This is due to the fact that the system needed to be maintained in order for the system to continue to remain useful to the end users. This includes releasing updates and patches to fix undiscovered errors in the earlier stages of the life cycle. Enhancements for the system's services may also be included as new requirements are discovered.

SYSTEM DESIGN

Chapter 4 – System Analysis

System analysis is the process of studying the design, specification, feasibility, cost, and also the implementation of a system for business use [29]. The phrase is the original definition for a system analysis, but for this paper, the system analysis would mean a process of gathering and interpreting facts, diagnosing problems and finally using the acquired information to improve the system further. This analysis is actually needed to get an overview of the system's requirements.

System analysis' purpose here is to determine the functional and non-functional requirements of this project. Besides the requirements, the programming language and hardware needs of the system are also determined during the system analysis.

The important buzzword for system analysis is 'Information Gathering'. This is vital as it provides the information needed to perform an analysis to find the above requirements. To carry out the information gathering, some approaches are made;

- Research

  Research is done through reviewing books and journals that contain the relevant information or discipline regarding the system. An example of a suitable place to conduct research would be in the Faculty's Document Room, where documentations by the past seniors are kept.

- Surfing the Internet

  Internet surfing is quick and quite efficient method to gather updated and useful information. The websites provide relevant information besides the expertise needed in developing this system. Some of the websites that had helped in the research of this system are listed in the reference section of this paper.

4.1 *Requirements*

The requirements for a system are determined soon after thoroughly studying and analyzing the systems objectives, scope, and also the existing systems (if any) or any other similar system in terms of concepts and execution. Requirements are essential as they provide the concrete ground from where the system will be based on and developed further until completion. In other words, this is the part where if the requirements are not defined properly, the system will not be able to fulfil the objectives and also the expectations from the end users.

Requirements can be roughly separated into two different types; functional requirements and non-functional requirements. Functional requirements capture the intended behaviour of the system [**30**]. This behaviour refers to the services, tasks or functions that the system is required to perform in order to complete its important tasks. As this type of requirement is important, any blunder here would hamper the system's objectives and for the worst case, redeveloping the system again to accommodate the appropriate functions.

The other type of requirement is the non-functional requirements. These requirements are generally the opposite of the above requirements; meaning they may or may not be part of the final set of requirements needed while developing the system. It is also a standard, so to speak, set by the developed system.

4.1.1 The Functional Requirements

- **Traffic Control Module**

  This module contains the main core of the system: the settings for the queuing disciplines can be manipulated and managed depending on the needs of the network administrator. These settings are conducted through a real-time connection to the designated server and the administrator is able to select the predetermined settings or if needed add more queuing disciplines to the module. The delete option is also available to remove unnecessary queuing disciplines in the module.

  Though I mentioned only queuing disciplines here, the classes and filters settings are also included as they are intertwined with each other, complementing each other to create the complete control over the network traffic (refer to Chapter 2).

- **Network Monitoring Module**

  Monitoring the network is just as important a process as setting the queuing discipline. It provides the administrator with valuable information about the current network's performance and usage first hand. The information gathered here would be used to create a suitable queuing discipline in the above module that corresponds to the networks current situation. This module includes the all-useful network graph with custom viewing options for the convenience of the network administrator.

- **User Management Module**

  Managing users is an integral part in maintaining the security in a system, or in other words privileges that are allowed to each individual while they are using the system. As the network administrator mainly uses this system, thus

this module would have only one level of access, which is the superuser or root. This is because only the superuser has the permissions to set the queuing disciplines in a network.

- **Report Generation Module**

  Reports are essential to a network administrator as it gives a broad overview of the entire networks condition, either current or past. This module would be using the data provided by the monitoring module (see above) to create reports, which could be customized by the network administrator according to what type of report that he or she needs.

- **Settings / Preferences Module**

  Every users preference is different and unique from each other. Thus it would be useful to the user if they had a choice in determining which type of setting that best suite their taste and style. This module would be functioning mainly on the settings for the interface part of the system e.g. fonts and also background of the web page.

4.1.2 The Non-Functional Requirements

- **Preciseness**

  For this system to perform up to the users' expectations, it must be accurate in providing up-to-date information on the network traffic. This is important as it influences the network administrator's decisions when determining the appropriate traffic control mechanism for the desired host. Any error in reporting the correct value could cause potentially disastrous problems to the entire network being controlled by the network administrator.

- **Flexibility**

  This system will be able to incorporate new technologies in the future because new technologies often crop up faster than ever in this evolving world of computing. The system will need to be flexible enough in order to accomplish this task.

- **Usability**

  The developed system needs to be user friendly, where the user will not have a hard time trying to understand the features in the system while he or she is using it. For this, the system should be easy to understand by the user in the shortest learning curve possible as not to hinder the user from losing interest in utilizing the system to its full potential. The ability to customize settings is also incorporated in the system to make the system more 'personalized' to the end user.

- **Modularity**

  Modularity involves breaking the programming into logical, manageable parts so that the distinct functions of the objects can be isolated from each other. This will eventually make the entire system more manageable and this inevitably leads to easier testing and maintenance efforts.

4.2 *System and Development Tools Used*

After reviewing and analyzing the list of requirements, the tools for developing the system will be decided. These tools are selected carefully as they influence the system's overall usability and in some cases, their performance. Thus, it is vital to understand the analysis previously discussed before deciding on the tools to be used for developing the system.

The following section discusses all the tools that will be used in the system.

4.2.1 Red Hat Linux 8.0

Red Hat Linux was chosen as the preferred platform due to its advantages compared to the other Linux distributions in the market. I will refer to an article in an Internet newsroom that talks about the Red Hat's position in the Linux distribution community [31]. They are listed below:

- Market Leader

  Currently, Red Hat Linux is the market leader, as seen from the MandrakeSoft website [32]. It shows very much the market is dominated by Red Hat, Mandrake, and also SuSE Linux, but above them all is Red Hat. Another point to make here is that, when most people think of Linux, they will most likely associate it with Red Hat, which is incidentally Red Hat is more like "the Microsoft of Linux".

- User Friendly Environment

  Though there are a lot of other Linux distributions in the market right now, some of them are not quite user friendly as Red Hat Linux. While perhaps Red Hat is not the first Linux distribution to be user friendly, it serves it's purpose well in trying to make life a little easier while using the Linux system for the first time.

- Certified Professionals

  Red Hat Linux is the de facto standard in the current Linux market. This can be shown in their own *Red Hat Certified Engineer* training programme, which was named top overall IT certification (Jan 2002) by Certification Magazine / Fairfield Research [33].

4.2.2 *Apache Web Server*

Apache was chosen from other types of web servers as it tightly integrated with the Linux operating system. Besides that, it is also the most used Web Server in the market [34] right now due to its stability and reliability in performing its tasks.

It has all the security needed and advanced features, and together with the ease of use and also the availability of documentation provided, this would seem to be the appropriate type of web server to be chosen over all. It is one of the many projects that the Apache Software Foundation is developing now.

4.2.3 *PHP Hypertext Preprocessor*

PHP is the programming language of choice as it is a cross platform language, and also the fact that PHP works well in the Apache web server itself. This reason is justified, as it is also another project of the Apache Software Foundation. This makes it appropriate to be chosen as it integrates well with the Apache web server, which was incidentally chosen as the preferred web server for the development of this system.

It is also a widely used general purpose scripting language as was written in the Netcraft survey [35].

4.2.4 *RRDtool*

RRDtool is actually a graphing tool that can be used for creating graphs and logs, which are useful when monitoring the network. RRD stands for Round Robin Database, where it is a system to store and display time-series data (i.e. network bandwidth, server load average, machine room temperature), stores data in a very

compact way, and also presents useful graphs by processing data to enforce a certain data density [36].

This tool was chosen as it works well with mrtg, also another graphing tool.

4.3 *System Requirement*

This project will be developed as a client server computing system. Three-tier client/server architecture is chosen over other architectures to implement this proposed system. Two=tier system is not suitable and feasible for this web-based system as it needs software to be installed on multiple clients. Due to this, three-tier architectures is more suitable as it requires several distributed machines for business, which is what this system will try to accomplish.

The benefits of three-tier client/server architecture are:

- Some upgrades can be done entirely at the server level
- Allows for component-based development, which in turn can increase reusability

The table below lists the minimum requirements for developing the project:

Component	Description
Microprocessor	Pentium II 166Mhz or above
RAM	At least 32 MB
Storage	At least 4 GB of hard disk space
Input devices	Mouse, keyboard
Output Devices	Printer
Operating System	Red Hat Linux 8.0
Video Monitor	EGA, VGA or compatible display

Table 4.1 – Minimum requirements for developing the Traffic Control System

Chapter 5 – System Design

System design is one of the important stages in the system development life cycle as it determines the success and usability of the entire system. Features of the system, the components or elements of a system, and also their appearances are described in the system specification. Requirements that were discovered in the analysis stage are the ones actually translated into these design specifications.

This chapter would discuss the system functionality design (hierarchy chart), and also a flow chart.

5.1 Hierarchy Chart

Hierarchy Chart is mainly used to identify the major modules that act on the data in a high-level picture of the system. The major functions are listed as the initial components of the hierarchy chart. These components are then broken down to sub-components (if available) and broken down to more sub-components if needed.

The Hierarchy chart at the next page shows the main modules relations with their respective sub-modules of the system.

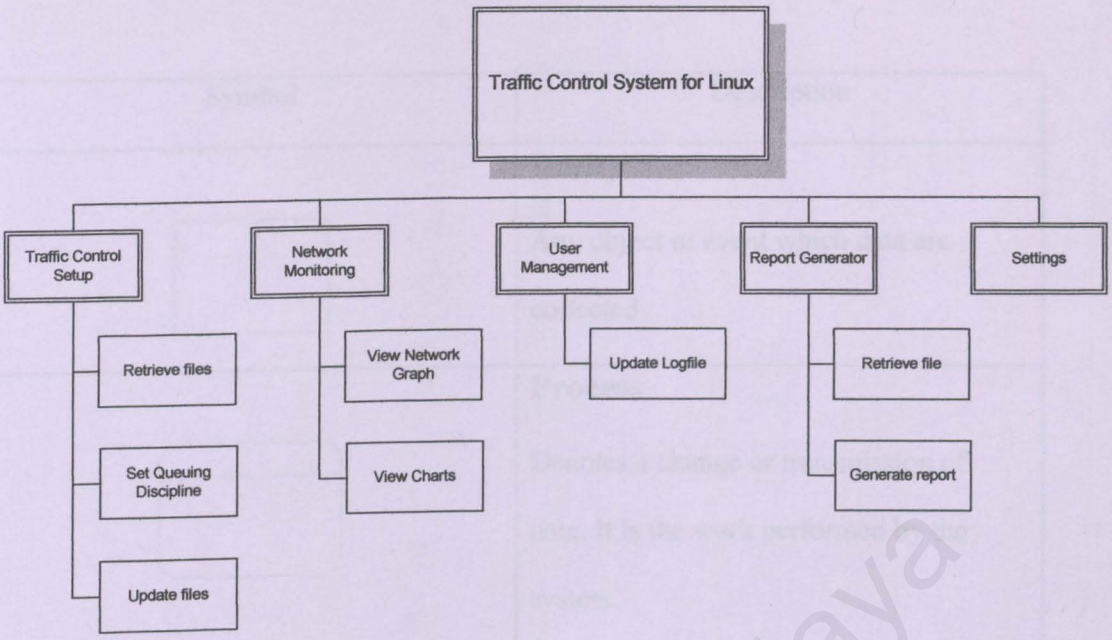


Figure 5.1 – Traffic Control system Hierarchy Chart

5.2 *Data Flow Diagram*

Data Flow Diagram is a graphical technique to represent information flow (information on the data moving from input to output)

This technique represents a system at any level of detail with graphical networks of symbols showing data flows, data stores, data processes, and also data sources. This data flow diagram is practically a network model of all possibilities with different details shown on different hierarchical levels. The process of representing different levels is called "levelling" or "partitioning" by some data flow diagram advocates.

Below are listed the basic symbols used in data flow diagrams together with their descriptions.

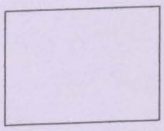
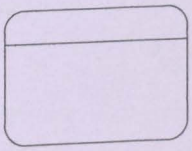
Symbol	Description
	**Entity** Any object or event which data are collected
	**Process** Denotes a change or transmission of data. It is the work performed by the system
	**Data Flow Direction** Referenced by a combination of the identifiers of the constructs that the data flow connects

Table 5.1 – Description of symbols commonly used in Data Flow Diagrams

5.2.1 Context Diagram

Context Diagram is the highest level in a data flow diagram and contains only one process, which represents the entire system. This diagram does not contain any data stores and is fairly simple to create once the external entities and the data flow to and from them are known.

Below is the context diagram for the Traffic Control system:

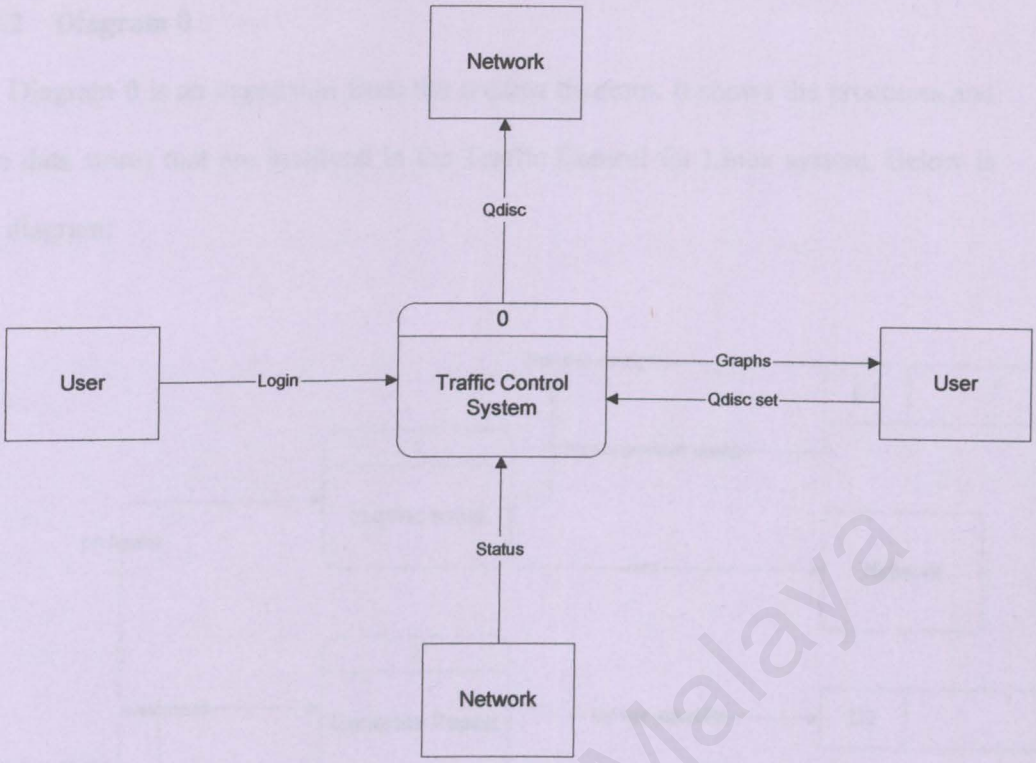


Figure 5.2 – Context Diagram for Traffic Control System

5.2.2 Diagram 0

Diagram 0 is an expansion from the context diagram. It shows the processes and also data stores that are involved in the Traffic Control for Linux system. Below is the diagram:

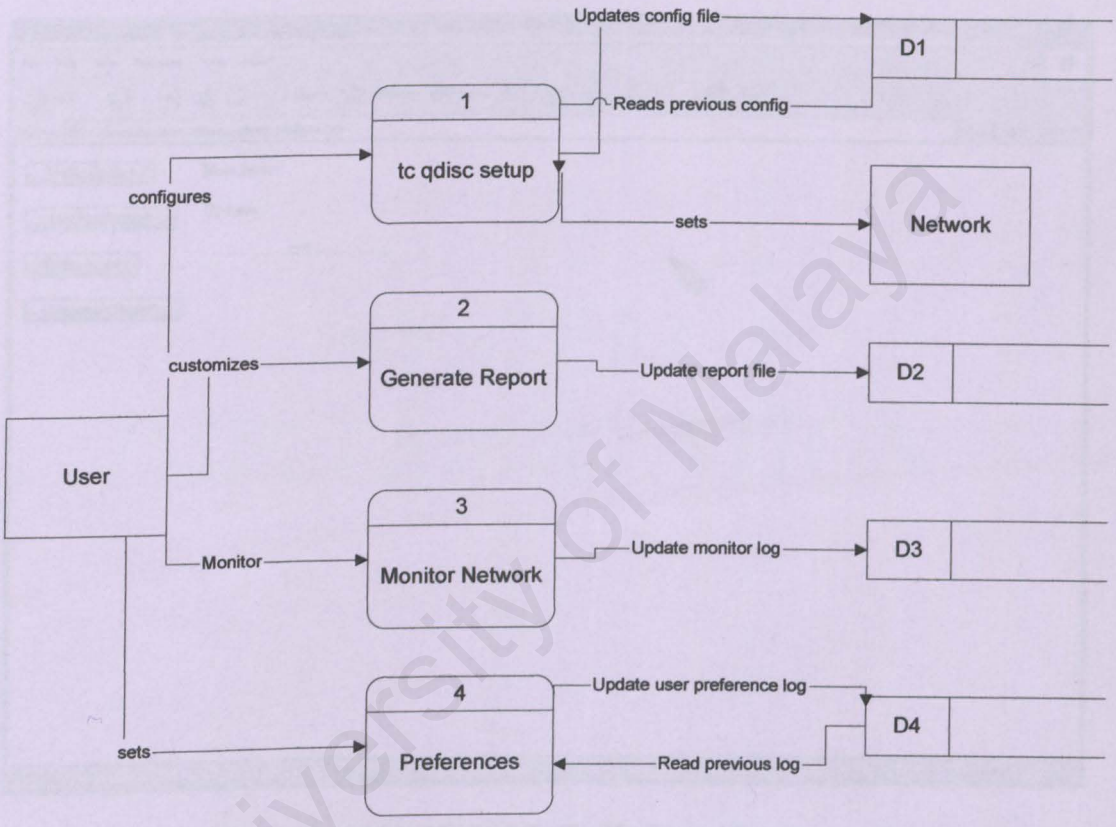


Figure 5.3 – Diagram 0 for Traffic Control System

5.3 *User Interface*

For this section I will design a prototype design of the main menu associated with the Traffic Control system. This will only be the draft outline of the web layout as there may be new functions to be added later on during the implementation phase.

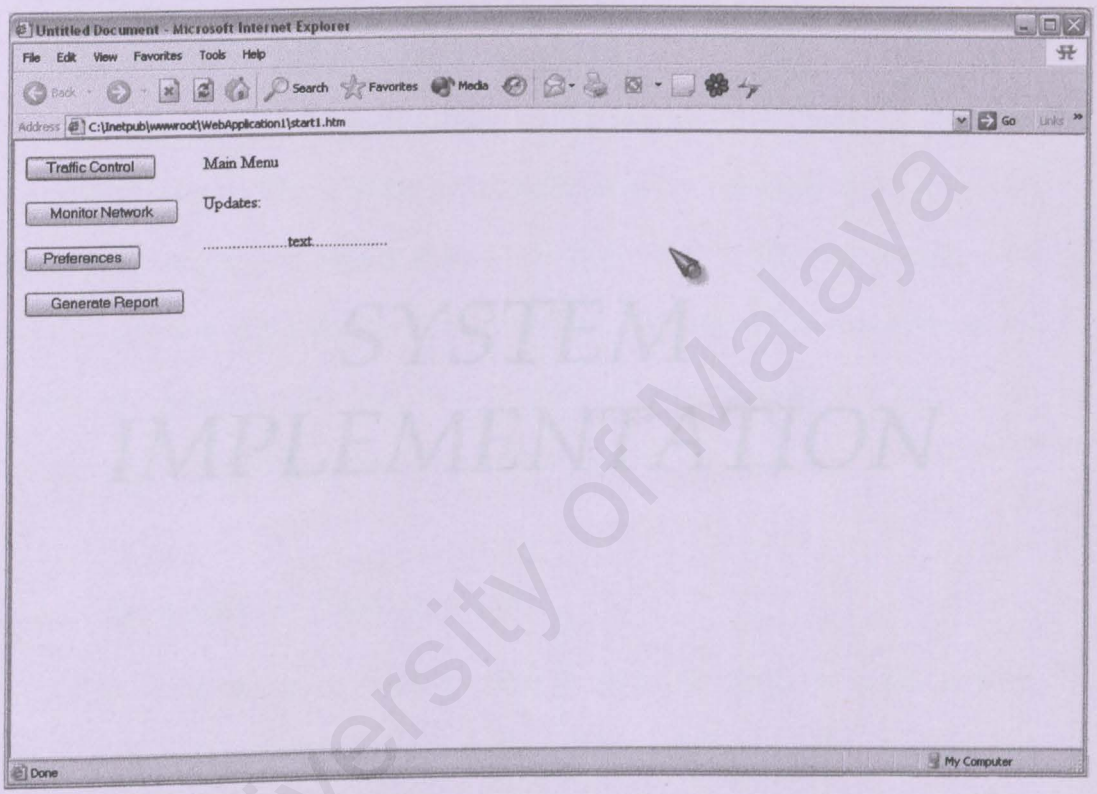


Figure 5.4 – Main Menu of the Traffic Control System

SYSTEM IMPLEMENTATION

Chapter 6 - System Implementation

6.1 *Introduction*

System implementation phase takes place after the system design phase. In the previous phases, all the foundation of the software has been laid out to clearly define the boundary of the system. This includes the literature review, requirements analysis, and also methodology and system design.

In this phase, the construction of the application is being put into the "production phase". This is where the designed modules and functions are integrated based on predetermined requirements. Steps involved in system implementation are building and testing (modules and sub-modules), moulding system requirements and design into program code. This leads to some software packages that needs to be installed for this implementation to take place properly.

6.2 *Development Environment*

The development environment plays an important role in the implementation phase as it is the place where the program either works or breaks. A stable and simple to manage development base may help to create a better system and may help to reduce the time needed to complete the project. Thus follows the hardware and software needed to develop the entire system of Traffic Control System:

6.2.1 Hardware Development Environment

The hardware configuration of the development system is listed below:-

- Intel Pentium Celeron 1.2 Ghz processor
- 80 GB hard disk drive
- 384 MB SD-RAM
- 16X speed DVD-ROM drive
- 15" Samsung Monitor
- Standard floppy drive, printer, and a broadband internet connection (Streamyx)

6.2.2 Software Development Environment

The software configuration for the development system is as follows:-

- Operating system: RedHat Linux 9 Personal Edition
- Web Server: Apache version-2.0.48
- Program coding:
  - User interface creation: HyperText Markup Language
  - Server side scripting: PHP version-4.3.4
  - Other languages: Bourne-again Shell (bash), C
- Browser: Mozilla Firebird version-0.7
- Web application development tools:
  - Text editor: kwrite, vi
  - Graphic manipulator: Adobe Photoshop 6.0

6.2.2.1 Operating System

Throughout the entire development of the system, RedHat Linux 9 was used as the platform for creating the software project. The latest release from RedHat Inc., it incorporates the optimized linux 2.4.20-8 kernel (customized by RedHat). A powerful and stable operating system, it is an open source software and thus it is free to use and install. Besides the aforementioned information above, it also works well with Apache web server and the PHP server side scripting language.

6.2.2.2 Apache version-2.0.48

The Apache web server, the most widely used web server in the Internet is a perfect environment to develop the system. This new version of Apache is the successor to the current Apache 1.3 version and though not recommended by the Apache developers to be deployed to mission-critical systems, it is still slowly being tested by various organizations as Apache version 2 has new features to support the ever evolving Internet technology. It also combines well with PHP to complete a stable web development platform.

6.2.2.3 Browser

The browser used is a revolutionary, new browser created by the well known Mozilla community. It is a no frills, fast, simple, and easy-to-use browser that can be enhanced and customized when the need arises. For this, I have chosen this browser for testing the web pages that I have created.

6.2.2.4 Web Application Development Tools

To write the coding for the system, a simple text editor is required. For this I have chosen kwrite, a software package included in the KDE desktop environment suite and vi, a shell-based text editor available in any Unix system.

As for graphics, I used Adobe Photoshop to manipulate the image to be inserted into the web system.

6.3 *Software Installation*

The installation part assumes that you have gotten all of the required software packages and are ready to install them. The software list is as follows, **RedHat Linux 9** distribution CD, **Apache 2.0.48**, **PHP-4.3.4**, **mrtg-2.10.13**, and **net-snmp-5.1.tar.gz**. All these steps do not require root privileges except for the **make install** part.

6.3.1 RedHat Linux 9

1. The easiest way to install is booting directly from the CD-ROM using the RedHat Linux 9 CD
2. Follow the instructions on screen and set the appropriate devices.
3. Select package to install (***Custom*** is recommended)
4. Select the partition to install Linux (using ***Disk Druid*** is recommended)
5. Setup the servers and network settings, and also a root password
6. Setup the bootloader (used when choosing which OS to boot – ***GRUB*** is recommended)
7. Choose the software packages to be installed
8. Install the operating system (no turning back!)

6.3.2 Apache version 2.0.48

1. Unpack the package (**tar –zxvf httpd-2.0.48.tar.gz**)
2. Enter to the created httpd folder (**cd httpd-2.0.48**)
3. Configure the makefile (**./configure --enable-so**), other options are available, type (**./configure --help**)
4. Compile the source (**make**)
5. Install the compiled source (**make install**). Defaults to /usr/local/apache2 if the path is not configured.

6.3.3 PHP version 4.3.4 (recommended to install apache first)

1. Unpack the package (**tar –zxvf php-4.3.4.tar.gz**)
2. Enter to the created php folder (**cd php-4.3.4**)
3. Configure the makefile (**./configure –with-apxs2=/usr/local/apache2/bin/apxs –with-mysql**), for other options (**./configure --help**)
4. Compile the source (**make**)
5. Install the compiled source (**make install**)
6. Copy the php.ini-dist file to /usr/local/lib/php.ini (**cp php.ini-dist /usr/local/lib/php.ini**)
7. Edit the php.ini file if necessary
8. Edit the *httpd.conf* file (location = **/usr/local/apache2/conf/httpd.conf**)
9. Check whether the path to the php module is installed

   LoadModule php4_module modules/libphp4.so

   Add these lines to it:

AddType application/x-httpd-php .php .phtml

AddType application/x-httpd-php-source .phps

10. Finally, start the apache server (**/usr/local/apache2/bin/apachectl start**)

6.3.4 MRTG version 2.10.13

1. Unpack the package (**tar –zxvf mrtg-2.10.13**)
2. Enter the created mrtg folder (**cd mrtg-2.10.13**)
3. Configure the makefile (**./configure**), for other options (**./configure --help**)
4. Compile the source (**make**)
5. Install the compiled source (**make install**), defaults to /usr/local/mrtg-2
6. To setup mrtg to monitor the traffic, create a *mrtg.cfg* file in the /etc directory (or anywhere you like)
7. A sample configuration is as follows:

```
WorkDir: /usr/local/apache2/htdocs/web/phtml/mrtg
WriteExpires: Yes
Interval: 5
Refresh: 300
Options[_]: growright, bits
Title[^]: Traffic Analysis for
#Other configuration options

Title[eric]: eric
WithPeak[eric]: wmy
MaxBytes[eric]: 125000
XZoom[eric]: 1.5
YZoom[eric]: 1.5
Target[eric]: 2:public@192.168.0.2
PageTop[eric]: <H1>Network traffic on eric </H1>
```

6.3.5 Net-SNMP version 5.1

1. Unpack the package (**tar –zxvf net-snmp-5.1.tar.gz**)
2. Enter the created net-snmp folder (**cd net-snmp-5.1**)
3. Configure the .config file (**./configure**)
4. Answer the questions during the configuration step (just press the RETURN key for the default value)
5. Compile the source (**make**)
6. Install the compiled source (**make install**), defaults to /usr/local/share/snmp
7. Setup the *snmpd.conf* file, located in /usr/local/share/snmp/snmpd.conf
8. A sample configuration is as follows:

```
rocommunity     public
rwcommunity     private
trapcommunity   trap_public
trapsink        eric
trap2sink       eric
syslocation     "Home Workstation"
syscontact      "enky82@yahoo.com"
# Net-SNMP-specific items
#keyw [args] limit(s)
load   5.0 6.0 7.0  # 1,5,15 load average max
disk   / 3%         # root filesystem < 3% free
proc   portmap 1 1 # Only 1 portmap process
proc   cron  1 1   # Only 1 cron process
procsendmail    # 1 or more sendmails
```

6.3.6 Other Requirements

6.3.6.1 Crontab

1. Cron is a scheduler program, for this system, it will be used to schedule the amount of time before polling the SNMP for network data
2. Edit the /etc/crontab file (must be root in order to write the file) using any text editor
3. Add the following line

   /5 * * * * root env LANG=C *<u>/path-to-mrtg /path-to-mrtg.cfg</u>

   /path-to-mrtg = path where the mrtg binary is located (usually /usr/local/mrtg-2/bin/mrtg)

   /path-to-mrtg.cfg = path where the *mrtg.cfg* file is located (depending on the above mrtg installation)
4. Save and exit the crontab file

6.3.6.2 tc

1. For the traffic control to work properly, the tc tool must be installed into the system. To check whether it is compiled already, type /sbin/tc. If there are some help items displayed then the package is already installed. If not, you would need to recompile the kernel to include the tc tool.
2. Recompiling the linux kernel requires a linux kernel source package. The latest version can be downloaded from the Internet, but the RedHat Linux 9 CD also includes the kernel

source. The steps listed here are for compiling a new (latest) kernel downloaded from the Internet

3. By default, for RedHat the linux kernel is located in /usr/src
4. Unpack the downloaded kernel source into the /usr/src directory
5. Remove the symbolic link (**rm linux-2.4**) and create a new symbolic link to the unpacked kernel (**ln –s /usr/src/linux-2.4-xx linux-2.4**, where xx refers to the version)
6. Enter into the linux-2.4 directory (it is a symbolic link to the real linux kernel source)
7. I would suggest using default configuration file to configure the kernel. Copy the ***.config*** file from the ***configs*** directory in the RedHat provided kernel source (**cp /usr/src/linux-2.4.20-8/configs/xx.config /usr/src/linux-2.4/.config**, where xx refers to the architecture of your system)
8. Enter into the linux-2.4 directory again and type **make xconfig.** This will configure the linux kernel using the graphical interface. You can also use **make menuconfig** or just **make config.**
9. The tc setting would be located under Networking Options. By default, it should have been selected. You may change any options that you might need.
10. Save and exit the kernel configuration.
11. Next run **make dep**. This will configure the dependencies needed by each package
12. Prepare the source tree for the build (**make clean**)

13. Change the version number of the custom kernel. It can be found in the Makefile in /usr/src/linux-2.4. Change it to any identifier you wish.
14. Compress the kernel to a boot image (**make bzImage**). It may take a while to finish.
15. If you have configured any modules for the kernel (most likely), type **make modules**
16. Next type **make modules_install.** This will install the modules in /lib/modules/<*kernelversion*>/kernel/drivers, *kernelversion* here refers to the number in the Makefile
17. Install the custom kernel and its associated files into the proper directories (**make install**)
18. The step above will copy the kernel into the /boot directory. The **initrd** image (used by modules) and **bzImage** are copied into the directory. The boot loader will also be modified accordingly but you may verify that they were done correctly. The bootloader file is usually in /etc/grub.conf (**GRUB**) or /etc/lilo.conf (**LILO**).

6.4 *Program Coding*

Since Traffic Control System main purpose is to control the network traffic from the web, it is classified under a web application. Thus, it utilizes web techniques to generate the appropriate content and functions required.

For the layout of the web page, HTML was chosen as it is the de-facto standard for the World Wide Web and it also conforms to international standards. Besides that, it is also easy to grasp and use. PHP is used for the server-side scripting needed to complete the main functions of the system, while the C language is used to enable the manipulation of the linux kernel permission control handles.

In addition, the shell scripting is used to execute command lines needed to run the tc program. This is essential as the PHP calls the shell scripts and executes the commands to enable traffic control to be implemented.

6.4.1 Coding Approach

Structured programming is a problem solving strategy as well as a type of programming methodology. The basic guidelines for a structured program are:

- The flow of control in the program should be as simple as possible
- The construction of the program should embody top-down design

Top-down design, also referred to as stepwise refinement is essential to the development of a well-structured program. This enables the programmer to 'see' the program in a clearer picture, thus able to decompose the problem into smaller parts in order to solve them.

While the top-down approach helps in creating a collection of small problems or tasks where each of these can be easily coded, a disciplined approach will Help in

improving program clarity and maintenance. This would ensure a readable code that in turn would less likely to contain mistakes.

The code that I have written is placed in similar folders; PHP scripts are placed in the **php** folder, design layouts are kept in **phtml** folder, CSS documents in the **theme** folder, and images are kept in the **images** folder. This is to ensure a modular code approach, which in turn is easier to debug.

An include file is also utilized in order to make the system easier to manage in case the original location of the Apache web server changes. The main reason for this would be the ease of changing paths without the painstaking steps of modifying every path in every file.

6.4.2 Login Approach

For this system, as it does not utilize any databases, the login process would need to be modified a little. The approach taken here is to match the password of the user against the UNIX passwd file located in **/etc/passwd**. As Linux incorporates most of the security implementations of the UNIX operating system, it would need a few tweaks in order to use this login approach.

RedHat utilizes the shadow password encryption technique to encrypt the passwords for the users of the system. The /etc/passwd file actually does not keep the encrypted value of the password; instead it is kept in the shadow file (/etc/shadow). Another catch is that the shadow file has the permission setting of 0400, meaning that only the superuser could read the file.

In order to bypass this security, a C binary program is needed. The C program's function will be to act as root temporarily to access the file and copy the entire contents of the file into a temporary location. Below is a snippet of the C code:

```
int main(int argc, char *argv[])
{
      FILE *pwdfile;
      FILE *tempfile;
      char line[MAXLINELENGTH-1];
      char *user;
      char *location;
      location = argv[1];
      setuid(0);

      pwdfile = fopen(PWFILE, "r");
      fgets (line,MAXLINELENGTH, pwdfile);
      tempfile = fopen(location, "w");

      while (!feof(pwdfile))
      {
            fprintf (tempfile, "%s", line);
            setuid(0);
            fgets (line,MAXLINELENGTH, pwdfile);
      }
      fclose(tempfile);
      fclose(pwdfile);
      return 0;
}
```

The setuid(0) line in the above snippet changes the current user of the C program to become root temporarily. The pointer location is an argument specified by the user running the program to write the entire contents of the /etc/shadow file.

Compiling this code, **"gcc getpasswd.c –o get"** creates a C binary program called get. To run it from the web page, the php code used is:

```
system ("/tmp/www/get /tmp/www/file.txt");
```

The above line simply runs the get program (located in /tmp/www) and supplying it with the location argument (in this case it's /tmp/www/file.txt). Next step is to make the file accessible by other users, thus another line of php code is needed. But this time the tasks are to change the permissions of the temporary file. The line added was:

```
system ("chmod 777 /tmp/www/file.txt");
```

The subsequent step to perform is to loop through the entire temporary shadow file and validate the login process. The function authenticate is listed below:

```
function authenticate($user, $pass)
   {
      $result = -1;

      $data = file("/tmp/www/file.txt");

      foreach ($data as $line)    // iterate through file
      {
         $arr = explode(':', $line);

         if ($arr{0} == $user)    // if username match test pwd
         {
            $salted = substr($arr{1}, 0, 12);  // get salt and
crypt()

            $enc_pw = crypt($pass, $salted);

            if ($arr{1} == $enc_pw)  // if match return 1
            {
               $result = 1;
               break;
            }
            else    // no match return 0
            {
               $result = 0;
               break;
            }
         }
      }
      return $result;    // return value
   }
```

The function takes two arguments, the username and password posted from the login form. The **foreach** line is used to loop through the file, separating each column to form arrays.

The **salted** variable is used to get the salt from the encrypted string in the temporary shadow file (for this example it takes the first 12 characters of the encrypted string). This salted value is then used to encrypt the password supplied by the user to create the **$enc_pw** variable. It is then compared with the password column of the shadow file. A return value is returned to the function and the value determines the validity of the login. The code is:

```
$status = authenticate($f_user, $f_pass);

system ("rm /tmp/www/file.txt");      // remove the file.txt

// if user/pass combo is correct
   if ($status == 1)
   {
      session_start();                          // session start
      session_register("SESSION");
      session_register("SESSION_UNAME");
      $SESSION_UNAME = $username;

      $file = "/main.php";
      $new_path = path_destination ($htmlpage_path, $file);
      header("Location: $new_path");
      exit();
   }
   else   // if check fails
   {
      $file = "/error.php";
      $new_path = path_destination ($htmlpage_path, $file);
      header("Location: $new_path?e=$status");
      exit();
   }
```

The return value is either "1" or "0" and the **status** variable is used to determine the user is a valid user or not. I have incorporated a secure feature where the duplicate shadow file is removed entirely from the system. This is to prevent any security compromises in case the hacker is able to hack into the web server.

The entire system also utilizes sessions in order to keep the system secure from being exploited. Sessions are registered from the moment the user logs in into the system, and for every page a session check is done before loading the page.

Below lists the session check code snippet:

```
   session_start();
   if(!session_is_registered("SESSION"))
   {
      // if session check fails, invoke error handler
      $file = "/error.php";
      $new_path = path_destination ($htmlpage_path, $file);
      header("Location: $new_path?e=2");
      exit();
   }
```

6.4.3 Controlling the traffic from the web page

The approach taken to implement traffic control is actually pretty straight forward. Values for the traffic control parameters are retrieved via text-boxes and also combo-boxes and processed by the relevant PHP scripts.

There is one requirement in order for the **tc** tool to be executed by other users besides root; the permissions for the **tc** binary has to be altered by the superuser to include the s-bit. From the shell terminal, the command entered would be "**chmod +s /sbin/tc**", where /sbin/tc refers to the path where the **tc** binary is kept.

The php folder keeps all the PHP scripts needed to perform the traffic control settings. All the scripts writes files to a temporary folder created under the /tmp/www directory, but again the superuser has to set permission rights for the /tmp/www directory in order for Apache to be able to write into it. Changing the ownership of the /tmp/www folder to nobody and also the chmod to 777 should be enough.

An example PHP script is shown below:

```
$filename = '/tmp/www/sfq_temp.txt';   // define file

      $handle = fopen ($filename, "w");

      $command = "#!/bin/bash\n\n";

   // insert tc commands to textfile
      if (isset($perturb) && isset($quantum) && isset($dev)) {
         $command = $command . "/sbin/tc qdisc del dev $dev root\n";
         $command = $command . "/sbin/tc qdisc add dev $dev root sfq
perturb $perturb quantum $quantum\n";
         fwrite ($handle, $command);

         fclose ($handle);

         system ("chmod +x /tmp/www/sfq_temp.txt");
         system ("/tmp/www/sfq_temp.txt");

         if (system) {              // just redirect it to another page
            $file = "/autonew.php";
            $new_path = path_destination ($htmlpage_path, $file);
            header("Location: $new_path");
         }
      }
```

The **handle** variable attempts to write or create a file in /tmp/www using the logic below it. The **command** variable holds the command lines to be inserted into the text file. After the commands have been written, the script immediately executes the created file in /tmp/www. The last few lines are just to redirect the page to another page after the command execution has been executed.

6.4.4 Validation approach

The validation approach taken uses only PHP for validating input from the user. Though it could be done similarly (and easily) with JavaScript, I opt to use PHP because perhaps the browser used may not have JavaScript enabled for security reasons.

Below is the sample validating script:

```
// check whether all fields are entered
   if (empty($dev) || empty($perturb) || empty($quantum)) {
      session_register('error');     // register an error handler
      $file = "/sfq.php";
      $new_path = path_destination ($qdisc_path, $file);
      header("Location: $new_path");
      exit();
   }
   else {
      ……functions here ….

   }
```

And the corresponding error handler in the page

```
<? if (session_is_registered('error')) echo '<h3
align="center">*** Error. Please fill in the empty fields below.
***</h3>'; session_unregister('error'); ?>
```

The sample validating script just checks whether the fields are empty, and if yes sends a session variable to the original page where the page will be refreshed

instantly together with the appending error message. Thus the user may not notice the slight difference when the page refreshes, as if it was done dynamically.

6.4.5 Determine paths to files

Installation paths may change depending on the user, thus it would be wise to incorporate the ease of changing the paths of the page links to be as easy as possible. For this I have created a function in the include file specifically for this task. Below lists the snippet of the code:

```
function path_destination ($path, $file)  //concatenate paths
{
   $path = $path.$file;
   return $path;

}
```

This function takes two arguments; the path variable defined in the include file and also the file name to be invoked. This function concatenates both of the strings to make a complete link to the specified page link.

6.4.6 Graph generation

The monitoring graphs are created using MRTG (Multi Router Traffic Grapher) written by Tobias Oetiker. For this system, I used MRTG to poll the snmpd daemon in order to gather the network traffic for each interface to be monitored.

The graphs are actually automatically generated by mrtg and updated every 5 minutes via the cron scheduler.

6.5 *Summary*

This chapter has given an insight into the system's implementation techniques and approaches to solving the requirements of the system. It covers most of the idea behind the coding as well as the coding itself. It also tells how the system is structured and thus making this system a little easier to comprehend.

It would also be noted that good design skill and structure are needed in order to make fewer coding mistakes during the implementation phase, which would ensure code readability and also modularity for easier management.

SYSTEM TESTING

Chapter 7 - System Testing

7.1 *Introduction*

Testing is an important step in the system development cycle. It is required to ensure that the system runs smoothly and according to its specification. The main reason testing is done is to find any flaws in the system. Software failures are defined as an occurrence whereby the system failed to function properly when it should be.

Each module and function in the Traffic Control System has been individually tested. After the modules and functions are completed, they are tested again (as a whole system) to make sure everything is in a working order.

In general, the Traffic Control System went through two stages of testing; function testing and also system testing.

7.2 *Types of Testing*

7.2.1 Function Testing

Function testing involves the testing of each working function as well as the working logic for each of the web pages. This is used to isolate the faulty functions from the working ones, thus simplifying the debugging task.

In short summary, the testing procedures in this type of testing is stated below:

- Testing the interface to make sure information flows in and out properly
- Testing the error paths to ensure it works properly
- Testing the conditional and non-conditional statements for errors

Throughout the entire development of the system, this function testing will carried out after any of the functions has been completed. This is to make that the errors are caught before they are links with other functions as to prevent a complex situation

whereby the spotting of errors will be harder than not. This testing will be done for each function until there are no more errors generated by the function.

7.2.2 System Testing

System testing is a more thorough testing procedure as it tests the entire system, including all the testing techniques of the function testing mentioned above. This will be used to verify that the system works seamlessly as a whole system as specified in the system and program design specification document. This is because the most common problems in large software systems are sub-system interface errors. Thus the sub-system testing should include more vigorous interface testing as to keep generating errors.

All link buttons were tested to ensure the hyperlinks were properly linked up. This is critical as a missing link may severely impeded the systems functionality and cause unexpected difficulties to the user of the system.

7.2.3 Example of a test case

In this system, if the user wishes to properly control the network traffic of the Local Area Network, he would need to enter the appropriate values into the textboxes and also may have to select the relevant values from the combo-boxes. Mandatory fields are properly aligned in a column for easy viewing and optional settings are also kept aligned in another column. Errors for not entering a value in the mandatory fields will be validated by PHP scripts but this system assumes that the user knows what values to enter into the text fields as a validation code is not present.

To properly configure the network, the user may choose from using a classless or classful choice. In this example I will show both of these choices to make a simple comparison between them.

No.	Test Procedure	Output/Error	Analysis of the test result and the corresponding solution
1	Click on the classful button (CBQ)	Links to the CBQ page	Hyperlink is working as expected
2	Fill in the values for every mandatory text box and click the *Continue* button	Links to the Class configuration page	Hyperlink is working well
3	Fill in the values for every mandatory text box and click the *End Configuration* button	Links to the TC configuration main page	The values are passed to the kernel for processing and the user is redirected to the TC configuration main page. Results of the configuration can be viewed from within the View Configuration link
4	Fill in the values for every mandatory text box and click the *Continue* or *End Configuration* button	Setting fail. Did not enter values into the mandatory fields. Error message written above the header of the page	Both buttons work properly. The main goal is to get the user to enter the mandatory values required by the kernel for processing the commands

5	Click on any link in the class configuration page	Links to another page (relevant to the name of the link)	Link is working fine. Links go exactly where the name of he link states
6	Click on the *Clear All* button	Any inputs in the textboxes are cleared	Button is working fine. All data was cleared and the page was reset
7	Fill in the textboxes and click the *Add to TC tree* button	Links to the class configuration main page.	Link is working fine. A session variable is registered for the corresponding page.
8	Fill in the textboxes and click the *Add to TC tree* button	Setting fails. Error message displayed on the page.	Link is working fine. User is not allowed to proceed unless he enters all the required fields
8	Click on the Update TC Configuration button	Links to the TC Configuration main page	Link is working as expected. Session variables are worked into a logic that executes scripts based on which session variable is generated
9	Click on the above navigation panel	Links refer to the correct pages	Link is working as expected. The logout function is executed properly

Table 7.1 – Classful configuration testing

As for the classless option, the logic behind the functions is the same as classful situations but the links are fewer. Below lists the table displaying the testing tasks being carried out:

No.	Test Procedure	Output/Error	Analysis of the test result and the corresponding solution
1	Click on the classless link (SFQ)	Links to the SFQ page	Links are working fine
2	Complete the form and click the *Update TC* button	No error, links to the TC configuration main page	Links are working fine. The values are passed to the kernel for processing via PHP
3	Complete the form and click the *Update TC* button	Setting fails. Error message is displayed in the web page	Links are working fine. The mandatory settings are supposed to be filled with the appropriate values
4	Click on the above navigation panel	Links refer to the correct pages	Link is working as expected. The logout function is executed properly

Table 7.2 – Classless configuration testing

7.3 *Full System Testing*

The main purpose of the function and system testing was to ensure that the code implements the design specified for the system. In this section, the testing objectives have a slight difference; to guarantee that the system does what the customer wants it to do.

This type of testing focuses on the system as a whole to ensure the software package performs to the users expectations. Activities include the system performance, security testing, configuration sensitivity, usability, data integrity, and error handling. Not all of these tests were carried out, but some of them are listed and

used here. This includes function testing, performance testing, and also security testing.

7.3.1 Function Testing

The full system testing begins with function testing, which is based on the systems functional requirements. Function testing is performed in a carefully controlled situation which would undoubtedly have a higher probability of detecting faulty code. Therefore a test should:

- Have a high probability of detecting a fault
- Know the expected outcomes and outputs
- Test both valid and invalid input
- Never modify the system to make the testing easier to perform

7.3.2 Performance Testing

This type of testing addresses the non-functional requirement of this system after the above testing is completed. System performance is measured using performance objectives set by potential users as they are the one who are going to utilize the system's functionality. During this testing, the effectiveness of the traffic control setting, viewing of the current **tc** configuration, as well as error handling are examined.

7.3.3 Security Testing

The security testing carried out attempts to test the vulnerability of the system in case an attack occurs. Prevention measures implemented (sessions) are utilized to make sure attackers do not easily get into the system due to the fact that some

program binaries are run as the superuser. Files that contain sensitive information are destroyed immediately after being used to prevent any garbage document which the attacking person would gain a chance to invade and compromise the system.

7.4 Summary

During this testing phase, several steps were performed. Function testing, system testing, and also a full system testing are planned and carried out accordingly. Function testing and also system testing focuses more on the functional verification of the components whereas the full system testing is designed to reveal bugs not related to independent components or units. It is used to validate the software once all of the smaller pieces of it has been combined. Three steps are included in this testing; they are function testing, performance testing, and finally security testing.

Testing is not only being carried out after the project has finished developing, it is an ongoing process that can start as early as the designing of the entire program structure. When an error is detected, a debugging process ensues to track down and eliminate the bugs.

SYSTEM EVALUATION

Chapter 8 - System Evaluation

8.1 *Introduction*

System evaluation is a process of evaluating the capability and usability of the entire system. Evaluation is shown as the final phase of the system development life cycle. This process involves several steps that include the identification of the systems strengths, system limitations and finally future enhancements. In this chapter also lists the knowledge gained and the problems encountered during the development of the system together with the corresponding solutions taken to overcome it.

8.2 *Problems encountered and Solutions*

During the implementation of the system, various problems were encountered. As a result, solutions to these problems were found and the details are highlighted in the following sections.

8.2.1 Lack of Mastery of Web Development Languages

The web development language that was used throughout the entire course of system implementation was PHP. Using this language is a challenging task as this is the first-time attempt to use and understand the language. Thus, the obvious solution to this problem would be to master the language as much as possible while at the same time coding the system.

References were also researched for from the Internet, and reference books were looked up in order to learn PHP as quick and easy as possible.

8.2.2 Lack of Mastery of the tc command

The tc command is the program being used to control the traffic in Linux. The mastery of this command ensures that the task of traffic control is made easier. As this command also requires a deep understanding of the underlying networking framework of the Linux system, the complex and somehow cryptic command line feature of the tc tool makes it all the more harder to get it working.

The solution for this was to look into the Internet for references and learning by trial-and-error technique.

8.3 *System Strengths*

8.3.1 No database

The absence of a database, the dynamic version that is, helps to keep the program small and manageable. This would favour the simple-yet-equally powerful software package using bare necessities to get the job done.

8.3.2 Security

The secure feature of the system, where the superuser-priviledged program binaries are run temporarily before being deleted, helps to keep doors closed on attempts to attack the web site.

8.3.3 Simple user interface

The simple user interface, devoid of most animations and pictures presents a rather easy to understand view of what the system does. It is also user-friendly whereby the user would not have any problems in getting around the web site to perform his or her intended tasks.

8.3.4 Online configuration features

The online configuration feature implemented on this system allows the user to control the network traffic of the LAN from anywhere in the world via an Internet access. This would free the user from needing to be at the physical location of the server in order to perform the same set of configuration rules.

8.4 System limitations

As in other systems, limitations are always present and for this system, it is also not an exception. These constraints though, can be further addressed and solved in future developments as well as system enhancements.

8.4.1 Variety

The amount of queuing disciplines available for configuration is barely enough. For future enhancements, the system could include more of the queuing disciplines to further add more variety into the configuration options. Thus the user would have more selections to choose from.

8.4.2 Graphing

The graphing features are not fully utilized in this version of the system. The graphs are enough to let the user monitor the traffic coming in and out of the monitored network interface, but lack the ability to gather more data to graph a more meaningful monitoring solution. For this, the future versions will need to incorporate this element into the system to further maximize the potential of it.

8.5 *Knowledge and Experience Gained*

Much knowledge was gained throughout the entire development of the system. Lots of exciting and interesting information and knowledge have been acquired, which would contribute to further enhance my knowledge of networking issues.

The operation of the server in Linux has brought about new challenges to overcome as it is still a very open operating system, with more potential to tap into. Learning the web techniques of the PHP language provides the knowledge that would be useful in the future, especially when the time comes to maybe create a web site of my own.

My personal skills were also not forgotten as there is room for improvement in learning to solve problems as well acquiring the skills to be able to work independently. Better writing skills, reports, and not forgetting user manual are also skills that I have learned during this short period of time.

8.6 *Summary*

In any are of software development, errors will always be part and parcel of the developing system. The problems were mentioned in section 8.2 were some of the problems encountered while implementing the system.

The system has also achieved the development objectives with the inclusion of the system strengths. However, the limitations mentioned previously are anticipated to be enhanced with a more powerful version of the working system in the future.

During the course of the system development, lots of useful knowledge and experience was gained. Knowledge of programming and designing were also improved.

8.7 Conclusion

This system has been successfully developed and the system objectives in the system proposal are also achieved. The modules and sub-modules are all integrated together to form a complete system capable of performing traffic control.

The system does not come without limitations though; In this current version, much is needed to further enhance the system with more powerful features. This creates an opportunity for the individual who is innovative and creative to further modify the system to match their own needs.

Throughout the system development, much knowledge and experience is gained. The technique on web-based programming, interface design, file manipulation, and programming and scripting languages were learned thus broaden the horizon of my knowledge skills.

REFERENCES

References

Books

[1] Parker, Timothy: Linux System Administrators Survival Guide, Second Edition: SAMS (2000)

From the Web

[2] Radhakrishnan, Saravanan: Linux Advanced Networking Overview (Implementation – Basic Principles) – 1999

[3] RFC 1633, Integrated Services

[4] RFC 2475, Differentiated Services

[5] "Linux Advanced Routing & Traffic Control HOWTO": Bert Hubert, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul B Schroeder, Jasper Spaans.: Published v0.9.0 Date: 2002/03/06

[6] Babcock, Michael T.: Linux QOS and TC – http://www.fibrespeed.net/%7Embabcock/linux/qos_tc/ (2000)

[7] Balliache, Leonardo: Practical QoS – http://www.opalsoft.net/qos/DS-21.htm

[8] Almesberger, Werner: Linux Network Traffic Control – Implementation Overview: Published April 23, 1999

[9] Stoica, Ian: Token Bucket Example, Carnegie Mellon University: http://www-2.cs.cmu.edu/~hzhang/15-744/token_bucket.pdf

[10] Simple End User Linux Project, http://www.seul.org/docs/whylinux.html

[11] linux4smallbiz.com: http://www.linux4smallbiz.com/guide/disadvantages (2002)

[12] http://www.hostsearchuk.com/q_nt.asp

[13] http://www.computersandinternet.com/july00-2.htm

[19] Web Server Concepts, Ford and Mason Ltd.: http://www.ford-mason.co.uk/resources/stw/node124.html

[20] Homepage of Apache Web Server: http://www.apache.org/foundation/faq.html

[21] Comparisons of the Five Server-Side Scripting Languages by Craig McElwee: http://www-106.ibm.com/developerworks/linux/library/l-script-survey/

[22] Amisu Salam-Alada, Abdul Waheed – Performance Engineering Laboratory, King Fahd University, Saudi Arabia: http://www.ccse.kfupm.edu.sa/~awaheed/pel/pel_2002_02.pdf

[23] Homepage of Traffic Discovery: http://tdisc.sourceforge.net/

[24] Perl Documentation Homepage: http://www.perldoc.com/perl5.6/pod/perl.html

[25] Homepage of Cricket: http://cricket.sourceforge.net/

[26] Extended Intelligence Inc. – Reuse-based Software Development Methodologies Explained: http://www.reusability.com/prod2.html

[27] Graphical Development Process Assistant Online: - Waterfall: http://www.informatik.uni-bremen.de/gdpa/def/def_w/WATERFALL.htm

[28] BridgeVIEW and LabVIEW: Professional G Developers Tools Reference Manual (1997,1998) National Instruments Corporation

[29] FOLDOC (Free OnLine Dictionary Of Computing) http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?system+analysis

[30] Functional Requirements and Use Cases (The Architecture Discipline) – Bredemeyer Consulting (1999,2000)

[31] http://www.newsfactor.com/perl/story/20036.html

[32] http://www.mandrakesoft.com/company/investors/bsa/position

[33] http://www.redhat.com/training/survey_fairfield.html

[34]

http://news.netcraft.com/archives/2003/09/01/september_2003_web_server_survey.html

[35] http://www.php.net/usage.php

[36] http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/

APPENDIX

Table of Contents

List of Figures

Introduction

Traffic Control System is a web based application for managing network traffic in a LAN. This system enables the user to easily control the network via a few settings in the web page as opposed to using SSH to control the remote server. This system bypasses the need to manually type all the commands in the terminal but instead focuses more to the GUI side.

The users who are allowed to use this system depends on the users in the server itself as the login script utilizes the server's /etc/passwd file. Thus to add or remove the users, the superuser of the server must manually add these users into the system in order for the user to be able to use the system.

About This Manual

The manual is organized into two parts, namely:

a) **Part 1 – Hardware and Software Requirements**

This section describes the requirements for both hardware and software for both the server and client computers

b) **Part 2 – System Walkthrough**

This section will attempt to guide the user through every step of using the system. Sample screenshots are provided for an effective explanation.

Part 1 – Hardware and Software Requirements

Below is the list of hardware and software requirements for the client and the server computer.

2.1 Hardware Requirement for Server

The following are the minimum hardware and OS configuration for the server

- 266 Mhz or higher x86-compatible CPU
- At least 128 MB of RAM
- 2.0 GB hard disk with a minimum of 500 MB free space
- Other standard peripherals, including keyboard, mouse, and monitor

2.2 Software Requirement for Server

The following are the software requirements for the server

- Red Hat Linux 9.0
- Net-snmp-5.1
- Mrtg-2.10.13
- PHP-4.3.4
- Apache-2.0.48
- Mozilla 1.5 or higher

2.3 Hardware requirement for Client

The following lists the minimum hardware requirements for the client computer

- At least Pentium 266 Mhz
- At least 64MB RAM
- Other standard peripherals, including keyboard, mouse, and monitor

2.4 Software requirement for Client

The following are software requirements for the client computer

- Windows 95/98, 2000, XP
- Internet Explorer 5.0. or later
- Netscape 7.0 and above

Part 2 – Using the system

2.1 Steps in using the system

1. First, open a web browser, using Netscape or Internet Explorer.
2. To access the login page, type the URL of the web site and press enter, When accessing through the Internet the address will be http://server's_ip_num/web/phtml/index.php, where the server's computer ip address is the server's location.
3. Figure 2.1 shows the login page for the system

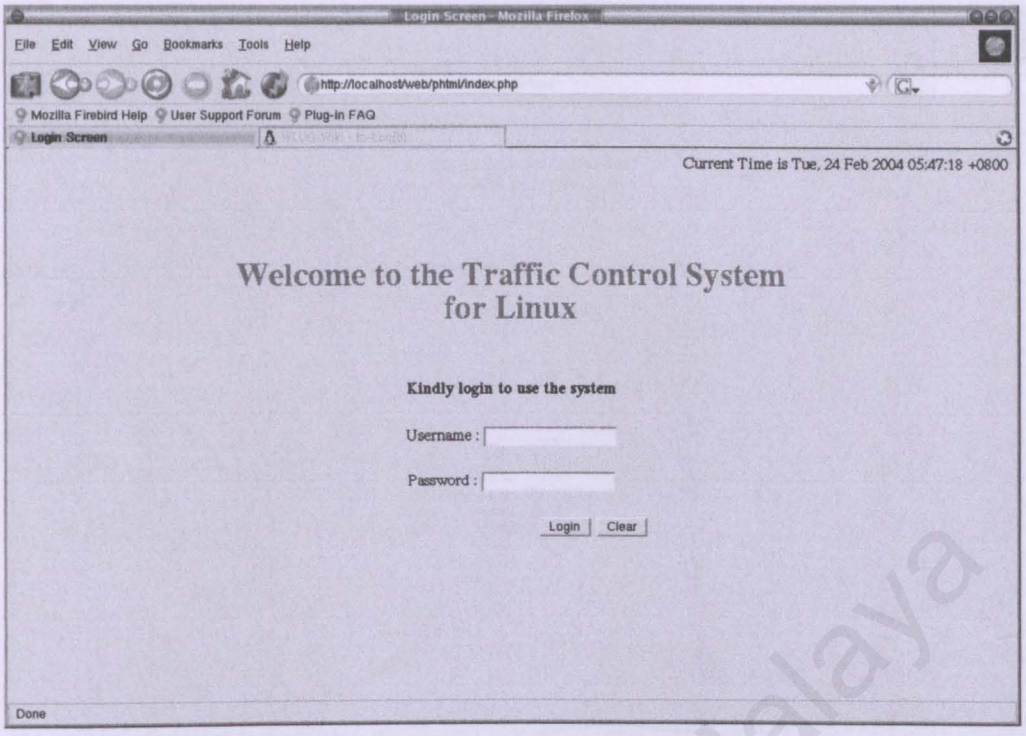


Figure 2.1 – Login page

4. After logging in, the main page will be displayed (Figure 2.3)

5. In case of error while logging into the system, an error page will be displayed (Figure 2.2)

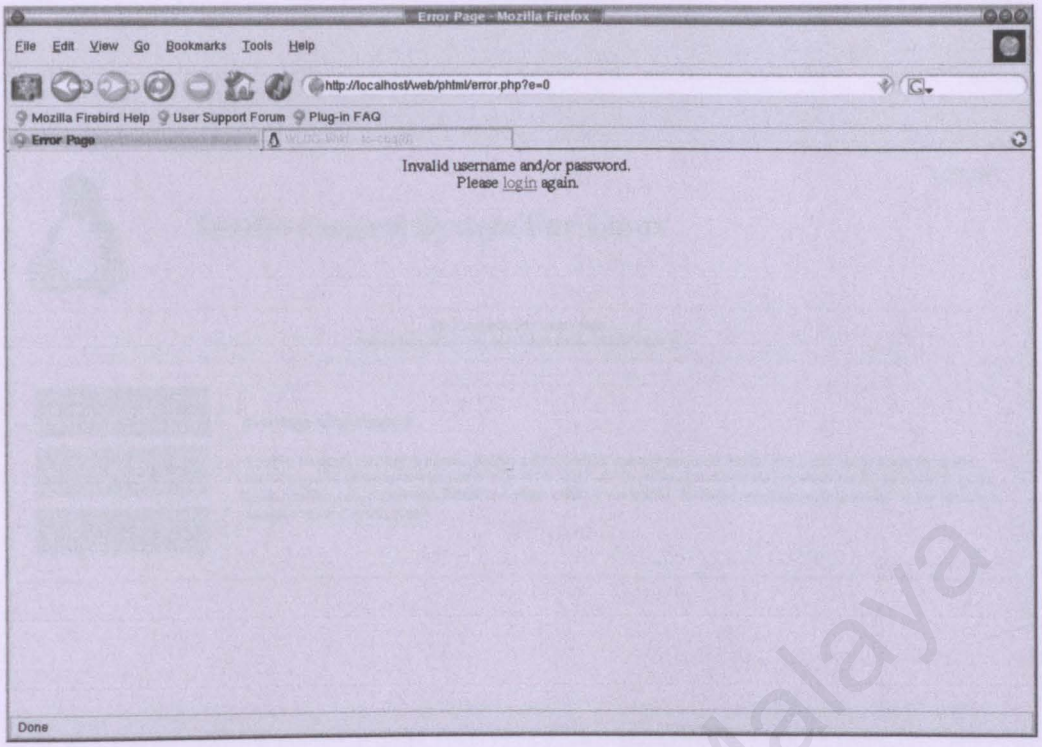


Figure 2.2 – Unauthorized login

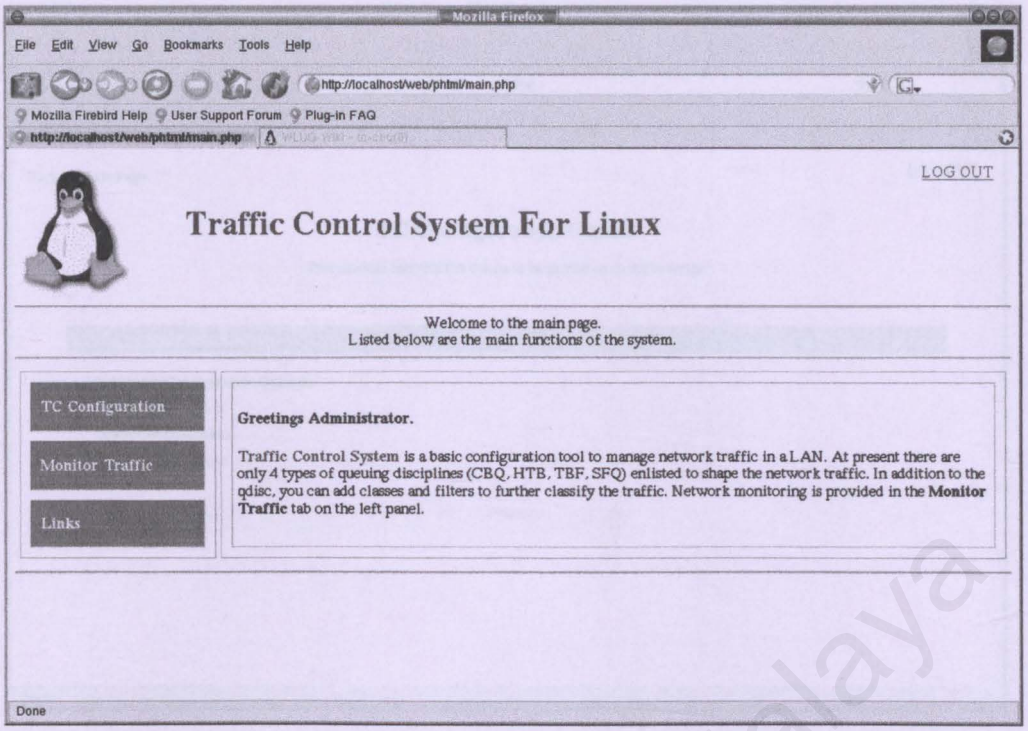


Figure 2.3 – Traffic Control System Main Page

6. Click on the TC configuration to link to the TC configuration page (Figure 2.4). This is the module where all functions controlling the network traffic is implemented
7. Clicking on the Monitor Traffic will lead you to the Monitor Traffic Page. (Figure 2.15)
8. Clicking on Links will lead you to the Links Page. (Figure 2.17)

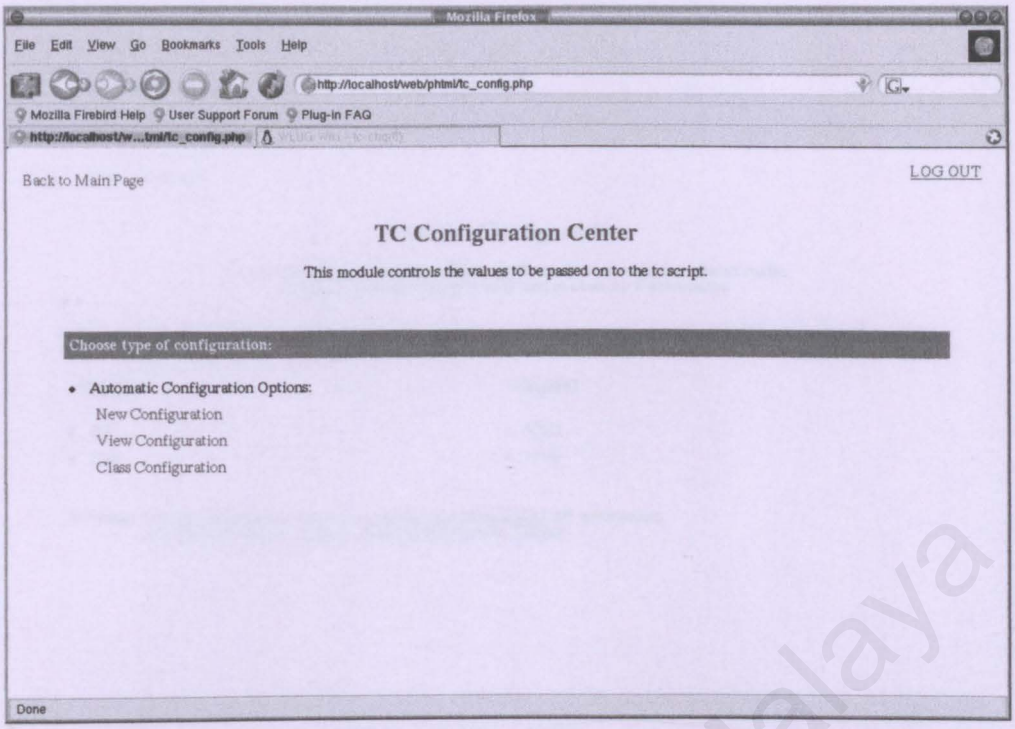


Figure 2.4 – TC configuration Main Page

9. New Configuration, creates a new tc configuration script. Options may vary. (Figure 2.5)
10. View Configuration, has options to view the current qdisc, class, and filter configuration status (Figure 2.13)
11. Class Configuration goes straight to the class configuration main page. (Figure 2.7). Classes must have the root qdisc set before they can be configured.

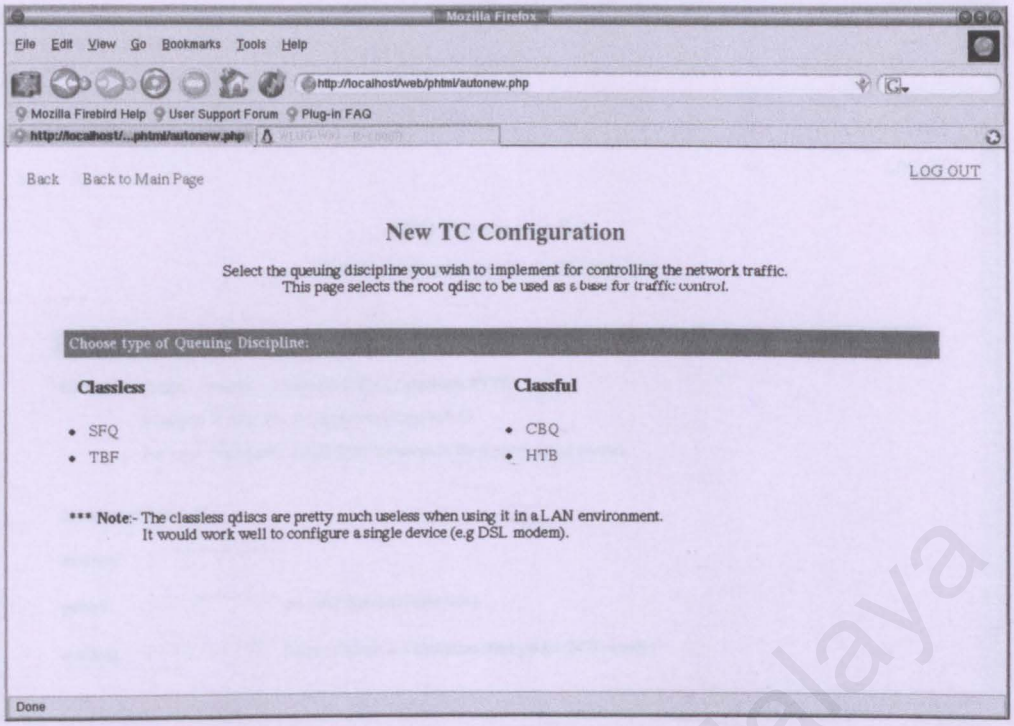


Figure 2.5 – New TC configuration Main Page

12. Classless and classful qdiscs each have two different kinds of qdisc. Classless qdisc are useful if controlling a single interface, while Classful qdisc are used to shape network traffic

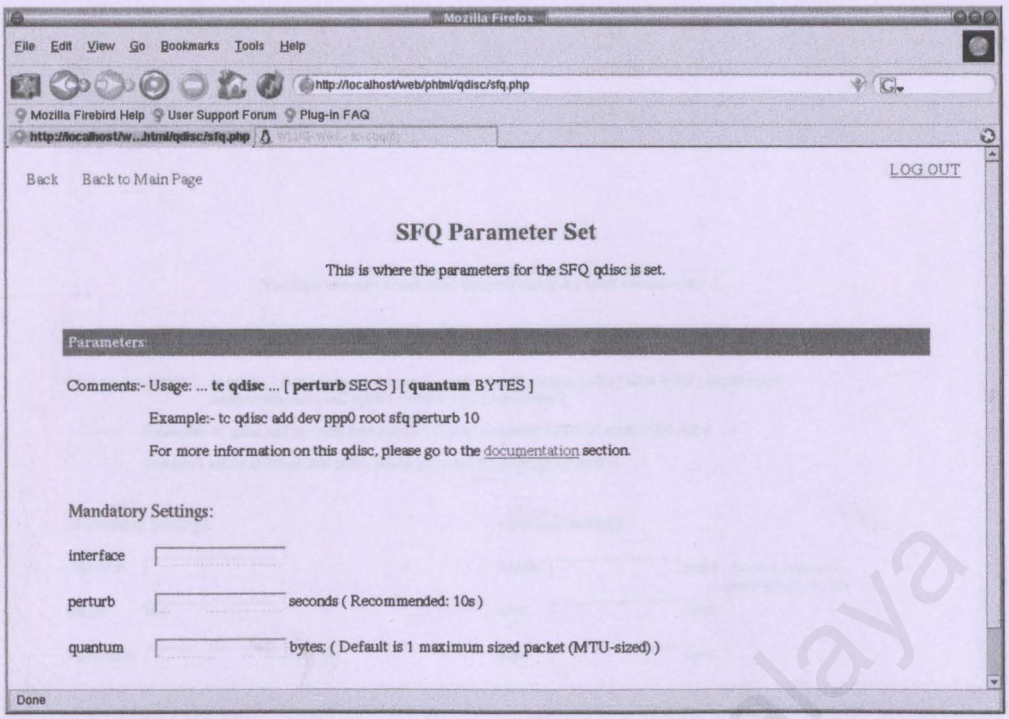


Figure 2.6 – SFQ parameter set (classless)

13. All the fields here are under the mandatory column, meaning you have to key in everything.

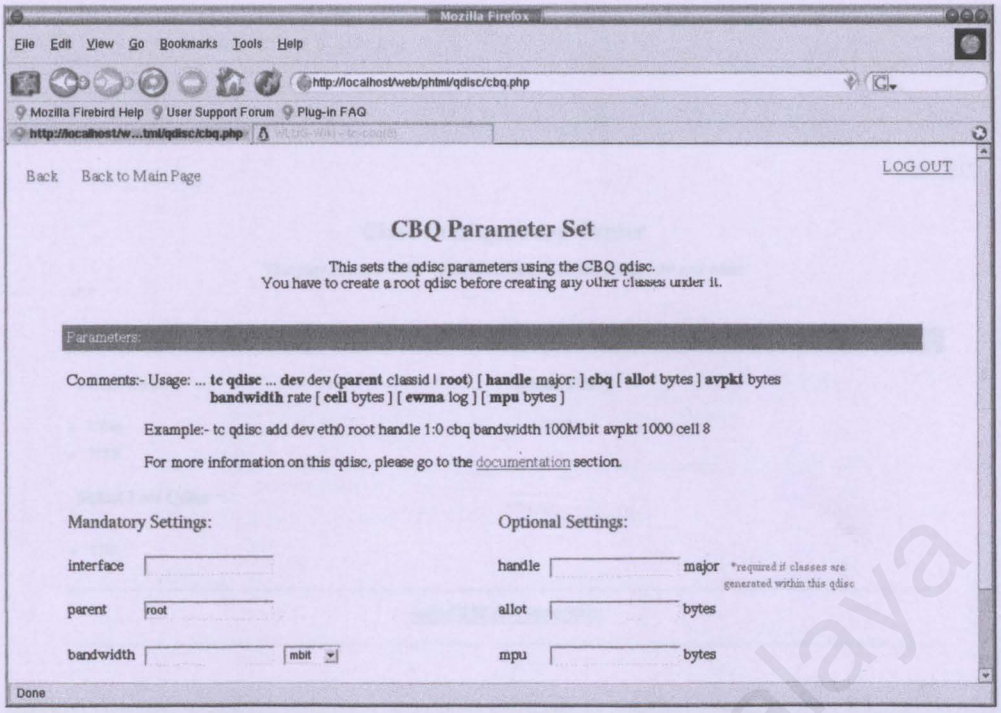


Figure 2.7 – CBQ parameter set (classful)

14. There are two different columns, mandatory and optional. Mandatory needs to be entered their values whereas optional is not necessary.

15. There are 3 buttons at the bottom, **Continue** – continue to create class (Figure 2.8), **End Configuration** – execute the current script, and **clear all** – clear every data from the textboxes.

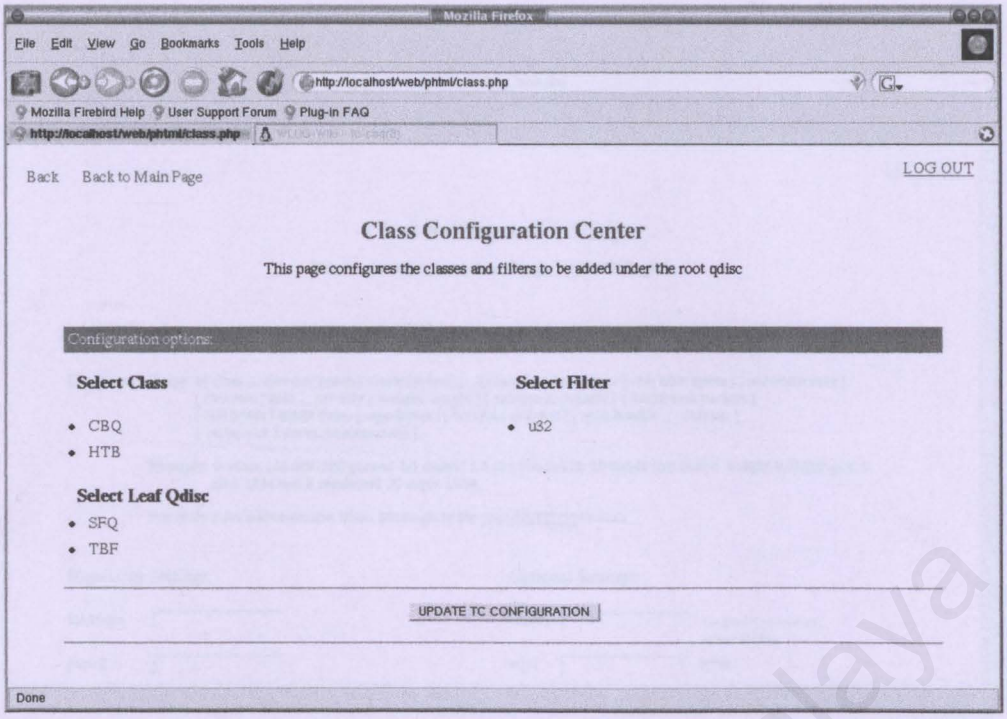


Figure 2.8 – Class setting Main page

16. Three options are provided, **Class** – generate class under the root qdisc, **Leaf Qdisc** – generate qdisc under the class handle, and **Filter** – create filter for traffic to be grouped to which class

17. The **Update TC Configuration** button will execute all the scripts that have been set previously

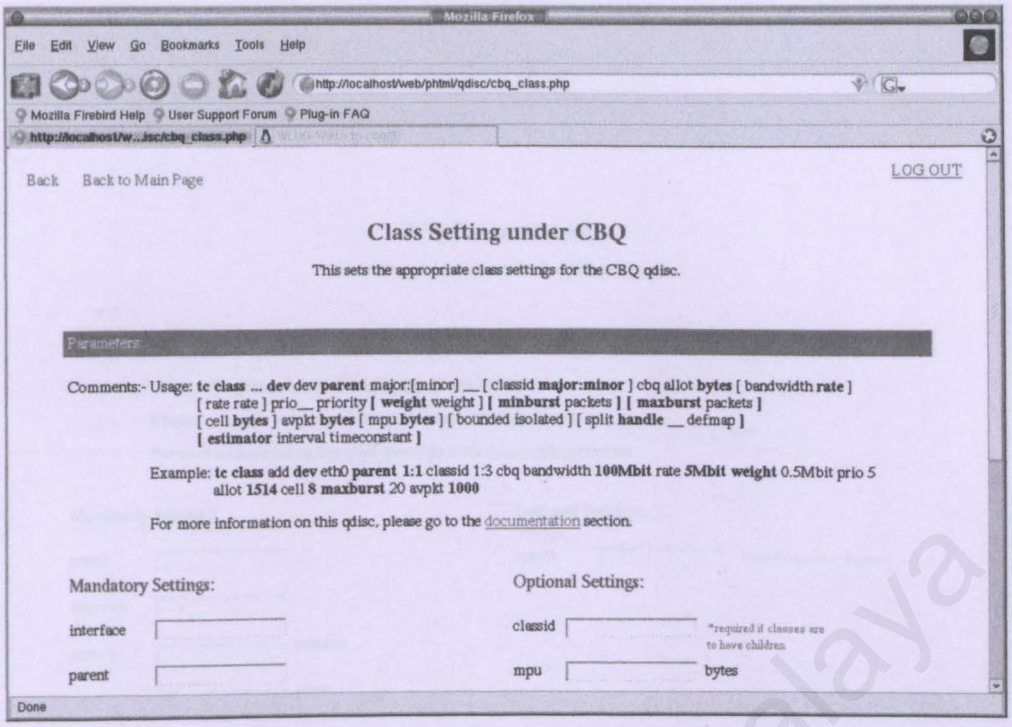


Figure 2.9 – Class setting using CBQ

18. Set the class under the CBQ root qdisc (if you used CBQ as root qdisc, you have to choose CBQ as the class also or else it will not work).

19. There are two buttons below, **Add to TC tree** – add the tc class command line into the script and redirect you to Class Setting Main Page (Figure 2.8), **Clear All** – clear all the values in the textboxes

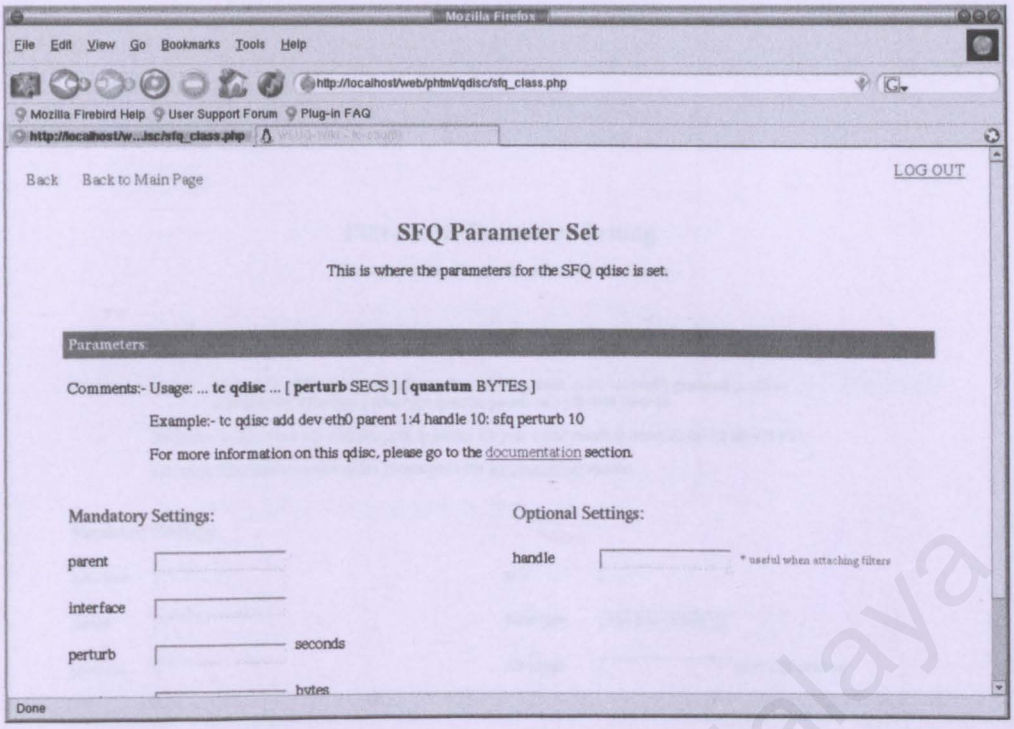


Figure 2.10 – Leaf Qdisc set

20. Sets the leaf qdisc under the class qdisc. Enter the required mandatory settings and click the **Add to TC Tree** button which will then redirect you to the Class Setting Main Page (Figure 2.8). The **Clear All** button is used to clear all the values in the textboxes

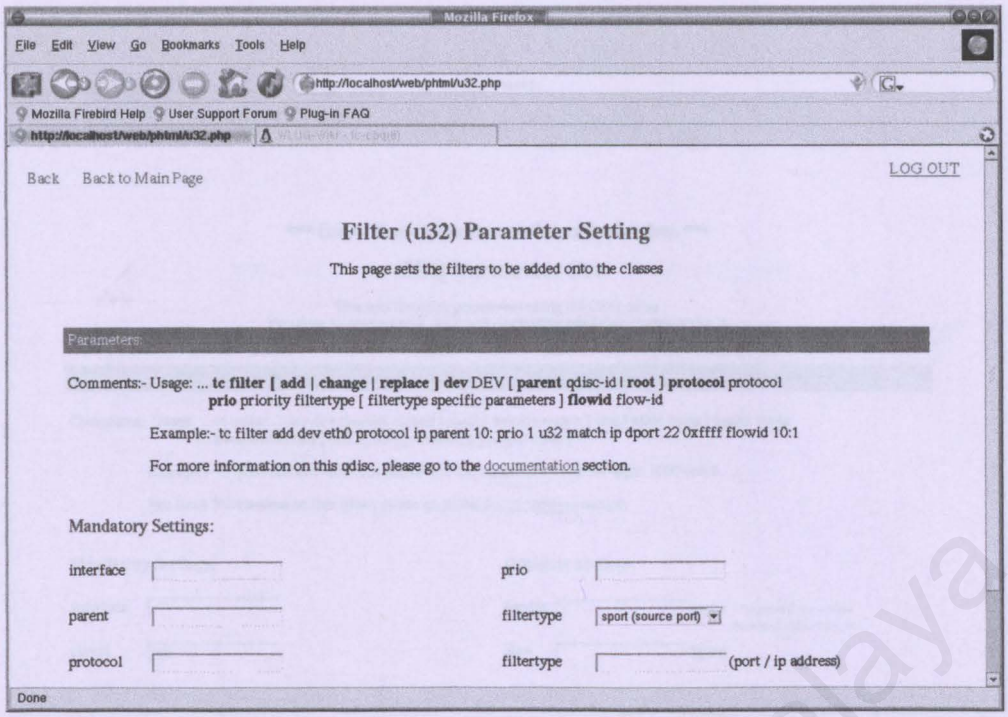


Figure 2.11 – u32 Filter set

21. Sets the filter settings to be appended to the class handle. Fill in the mandatory settings and click **Add to TC Tree** will redirect you to the Class Setting Main Page (Figure 2.8). **Clear All** will remove all the values in the textboxes.

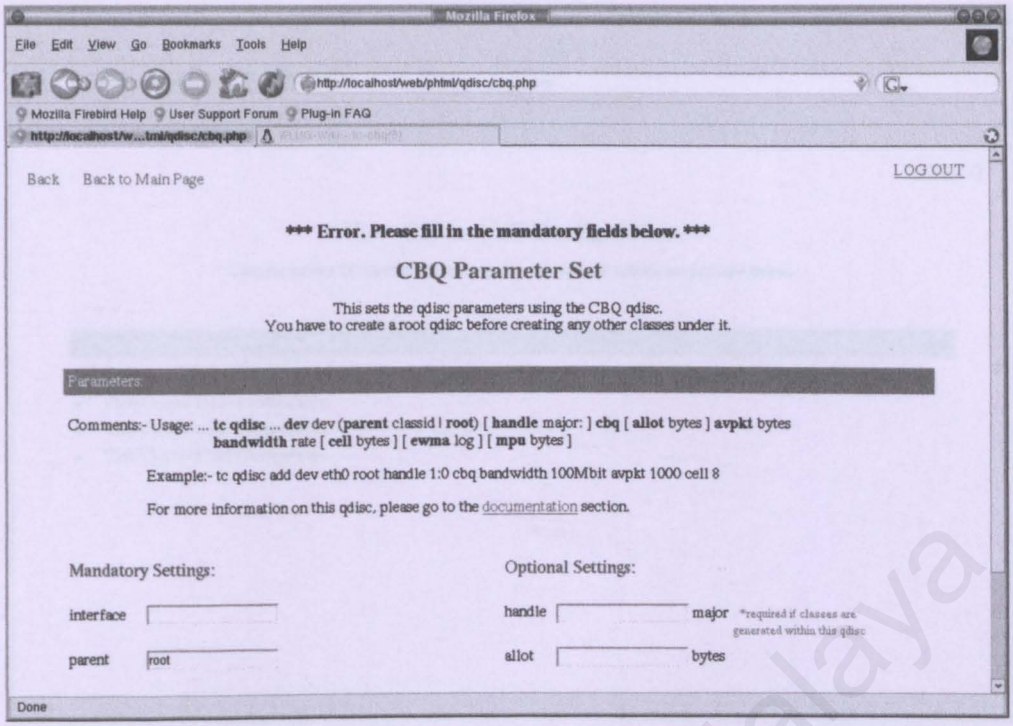


Figure 2.12 – Error handler example

22. If you did not enter any value in any of the mandatory textboxes, an error message will be displayed on top of the page.

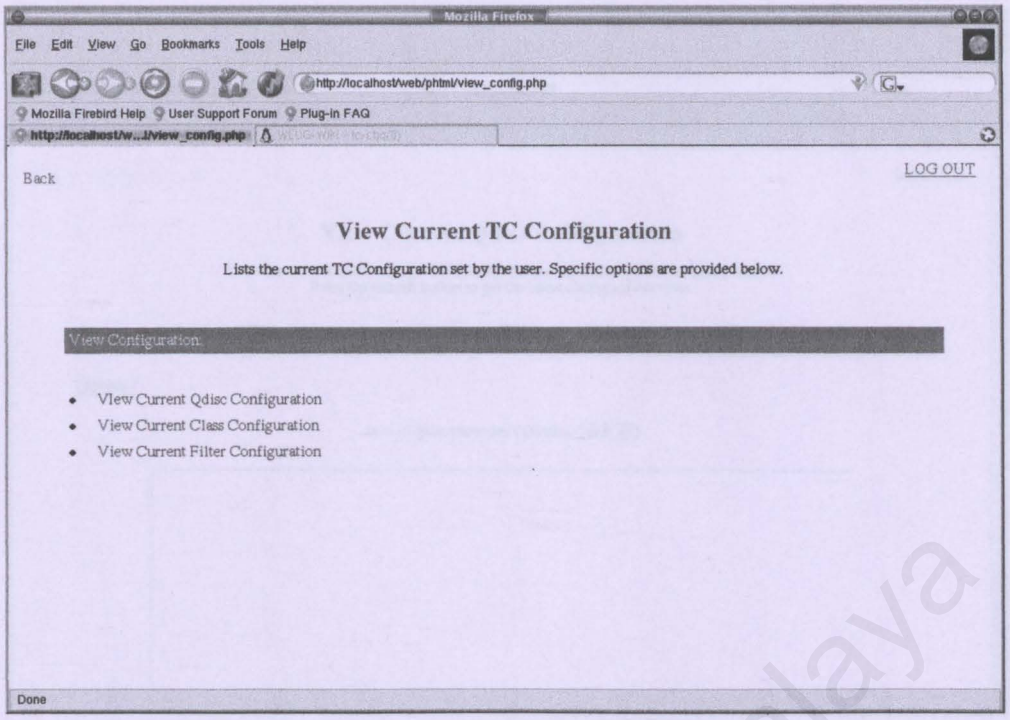


Figure 2.13 – View Current TC Configuration Main Page

23. Click on the links to view each of the configuration settings. An example is shown in Figure 2.14

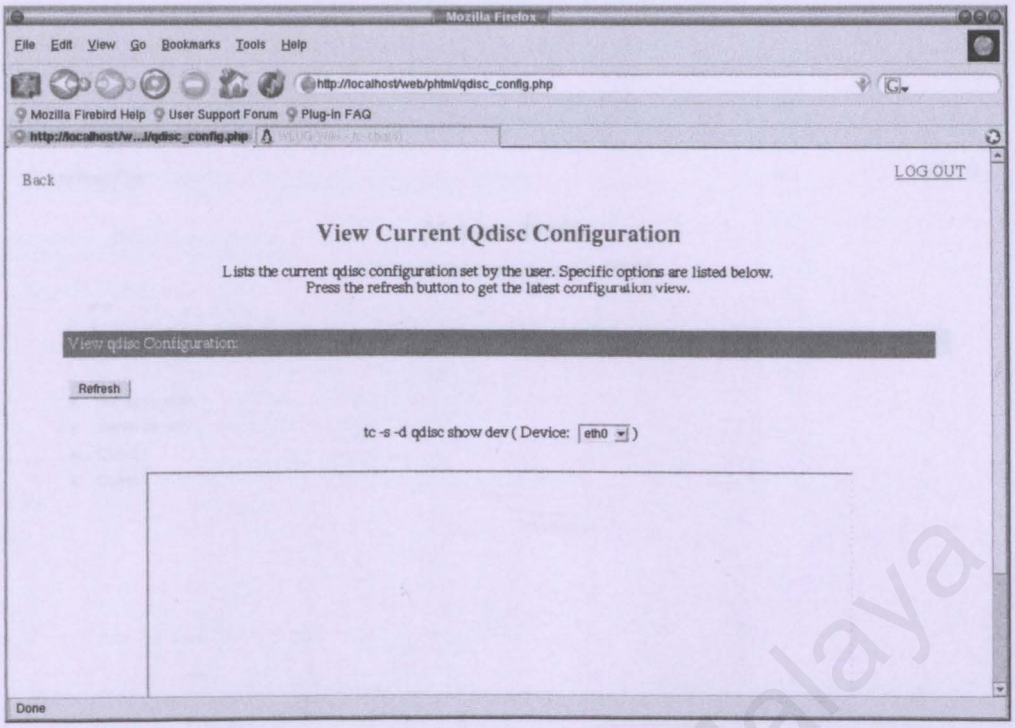


Figure 2.14 – View Current TC Configuration example

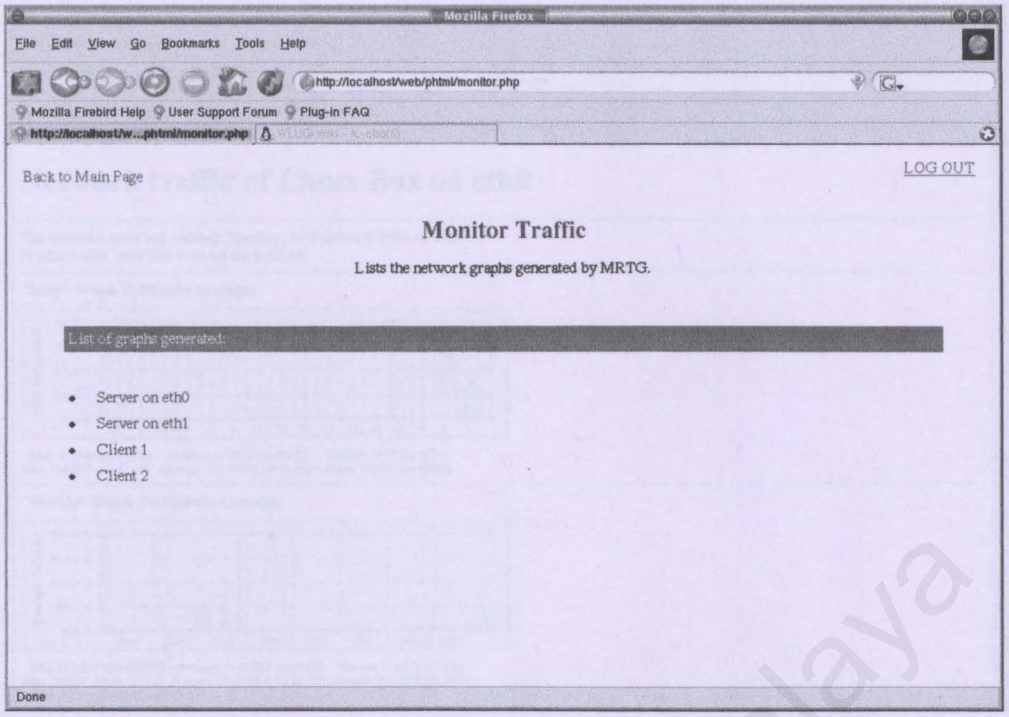


Figure 2.15 – Monitor Traffic page

24. Click on any of the links to view their specific graphs. An example graph is displayed in Figure 2.14

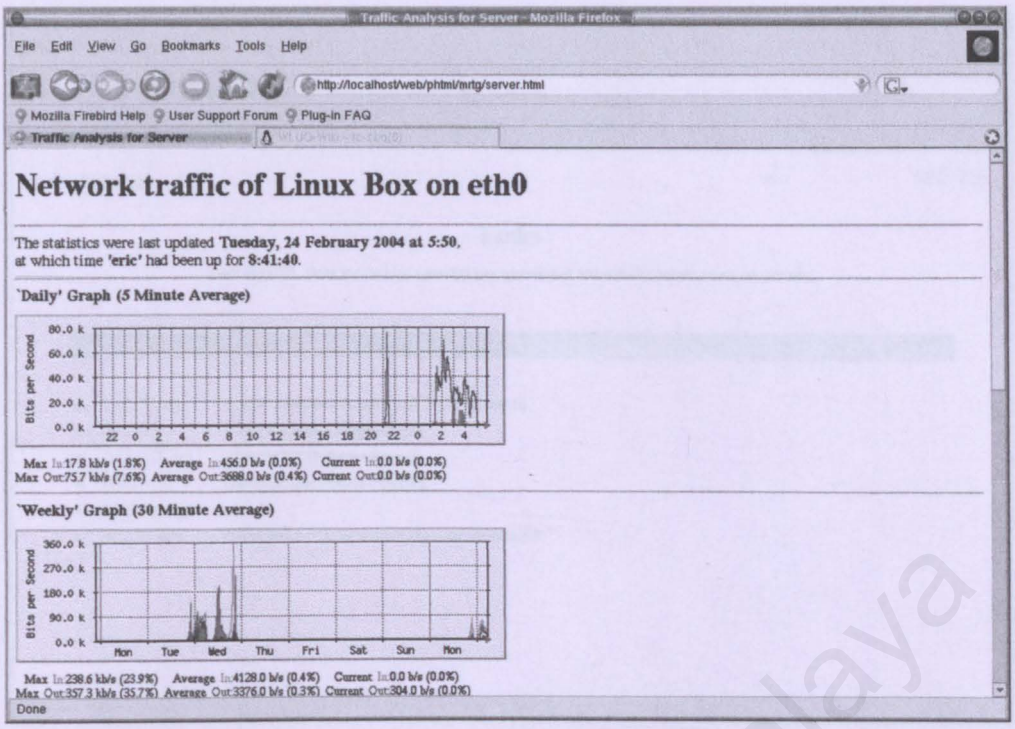


Figure 2.16 – Monitor Traffic page example

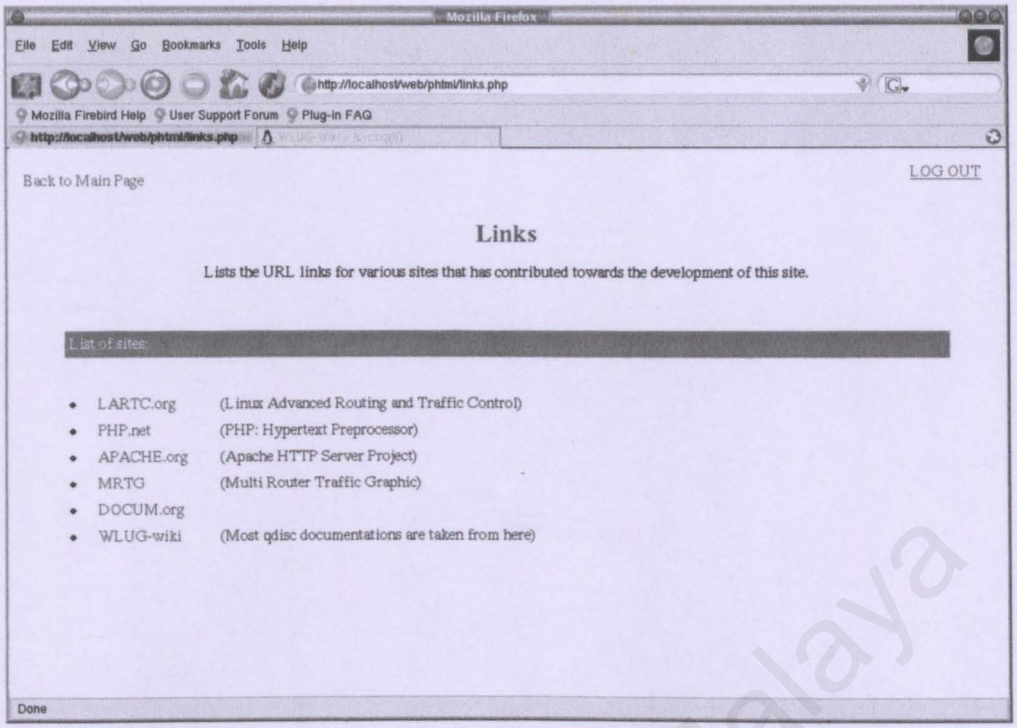


Figure 2.17 – Links

25. Click on any of the hyperlinks above to their respective URL destinations